Tampereen yliopisto
Tampere University

TIE-02306 ”ItSE”

Introduction to Software Engineering

5 credit units

07-lifecycle-ItSE2019-v5

Tampereen yliopisto
Tampere University

Course contents (plan)

1. Course basics, intro
2. Sw Eng in general, overview
3. Requirements
4. Different software systems
5. Basic UML Diagrams (”Class”, Use Case, Navigation)
6. UML diagrams, in more detail
7. **Life Cycle models**
8. Quality and Testing
9. Project work
10. Project management
11. Open source, APIs, IPR
12. Embedded systems
13. Recap

7. Life Cycle models (etc.)

- software life-cycle
- software development life-cycle (SDLC)
- different models
  - waterfall
  - spiral
  - iterative
  - incremental
  - agile (e.g. Scrum) and Scrum-BUT Scrum in more detail
  - XP (eXtreme Programming)
  - lean
  - DevOps
  - SAFe
  - kanban
  - PRINCE2
  - hybrid models…
- GSD = global sw dev, distributed sw dev, outsourcing, off-shoring
- maintenance

Current at course (w 43)

- we have eight project groups left
- **no WEs this week (1st presentations; Thu 24.10. two sessions)**
- after those we think next week WE7 (dev. processes) groups
- **continue updating your Trello (kanban) boards** = use at your process

***** remember EXAM 1/3 (weeks 41-43) *****

- **results will be published after week 43**
- ***remember EXAM 2/3 (weeks 44-46); diagrams (Dia or EA tool)***
- **check that you can log into PRP system (link via Moodle)**

Remember InnoEvent (04-08.11.2019) www.innoevent.fi

Backlog items with deadline

- **09.09.2019** at 23:59 Group forming (Moodle)
- **15.09.2019** at 23:59 Trello creation (Trello)
- **13.10.2019** at 23:59 Phase 1 documentation (Moodle)
- **13.10.2019** at 23:59 Phase 1 presentation slides (PRP-tool)
- **Week 43** Phase 1 presentations (Physical realm)
- **03.11.2019** at 23:59 Phase 1 peer feedback (PRP-tool)
- **17.11.2019** at 23:59 Phase 2 documentation (Moodle)
- **17.11.2019** at 23:59 Phase 2 presentation slides (PRP-tool)
- **Week 47** Phase 2 presentation (Physical realm)
- **01.12.2019** at 23:59 Phase 2 peer feedback (PRP-tool)
- **08.12.2019** at 23:59 Final delivery of project documentation (Moodle)
- **15.12.2019** at 23:59 Final peer feedback and self assessments (PRP-tool).

Tampereen yliopisto
Tampere University

Weekly exercise attendees

	w36 WE1	w37 WE2	w38 WE3	w39 WE4	w40 WE5	w41 WE6	w44 WE7	w45 WE8	w46 WE9	w48 WE10
WED	0	14	9	5	8	9				
THU	21	13	14	17	16	13				

We will continue two Weekly Exercise groups, as long as the number of attendees are reasonable.

Tampereen yliopisto
Tampere University

Now, the main contents today is methods, methodologies, processes and technologies...
most emphasis on agile and Scrum.

Tampereen yliopisto
Tampere University

One early (1950..1960) way of describing program development

The first mentioning of **waterfall model**:
Herbert D. Benington, Symposium on advanced programming methods for digital computers, 1956.

The first formal description:
Winston W. Royce, "Managing the Development of Large Software Systems", 1970.

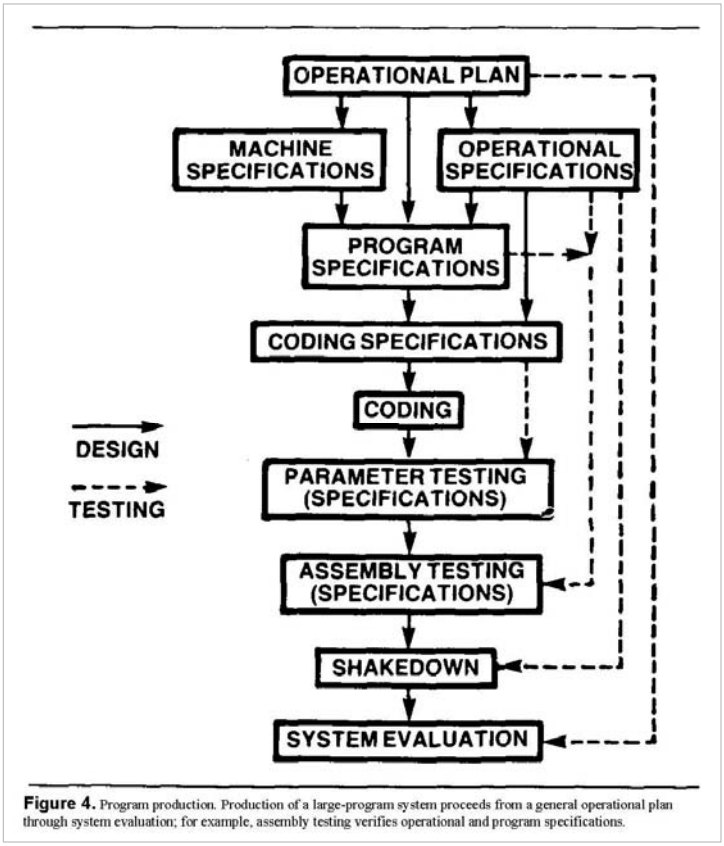


Figure 4. Program production. Production of a large-program system proceeds from a general operational plan through system evaluation; for example, assembly testing verifies operational and program specifications.

Why methods;

Software systems became built more and more, so defining some solid way of working made life a bit easier.

SLC = software life cycle

software life cycle. The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. *Note:* These phases may overlap or be performed iteratively. *Contrast with:* **software development cycle.**

In **every** Sw Eng project there is some
- **preliminary analysis**
- **requirements specification**
- **design**
- **implementation**
- **testing**
- **documentation**.

But methods and processes, way of working, have changed.

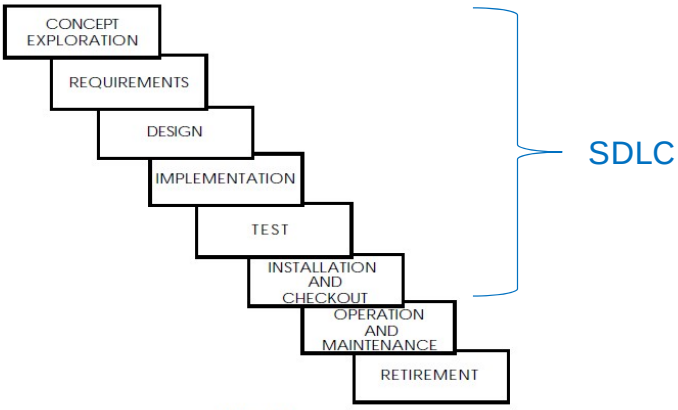


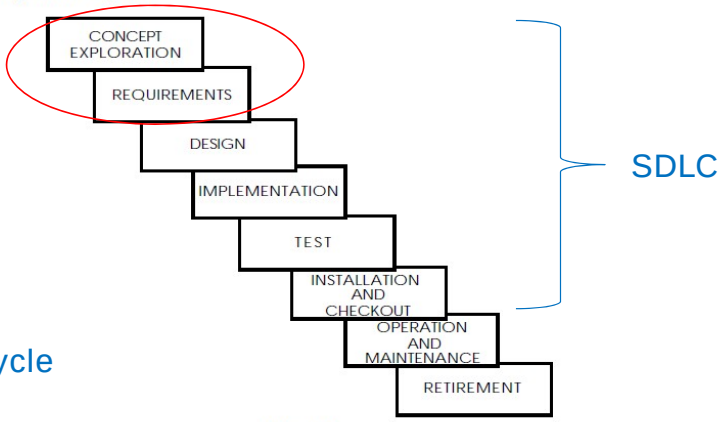
Fig 15
Sample Software Life Cycle

[**IEEE Standard Glossary of Software Engineering Terminology,** Std 610.12-1990(R2002)]

SLC = software life cycle

software life cycle. The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. *Note:* These phases may overlap or be performed iteratively. *Contrast with:* **software development cycle.**

TIE-02306
"ItSE" is about
that small part



SDLC = software development life cycle

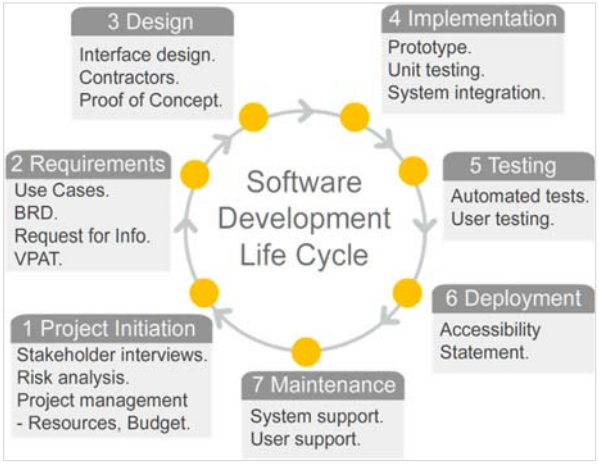
Fig 15
Sample Software Life Cycle

[**IEEE Standard Glossary of Software Engineering Terminology,** Std 610.12-1990(R2002)]

“Waterfall” shaped as a SDLC circle

Well, in **every** Sw Eng project there is some
- **preliminary analysis**
- **requirements specification**
- **design**
- **implementation**
- **testing**
- **documentation**.

[Husson University]



[University Of Melbourne]

In every life cycle model there are the same basic items in some role.

BRD= business requirements document
VPAT= voluntary product accessibility template.

Tampereen yliopisto
Tampere University

Software life cycle

ISO/IEC/IEEE 24765:2017(E)
Systems and software engineering — Vocabulary
Second edition, 2017

3.3803
software development cycle
SDC = SDLC
1. period of time that begins with the decision to develop a software product and ends when the software is delivered
cf. software life cycle

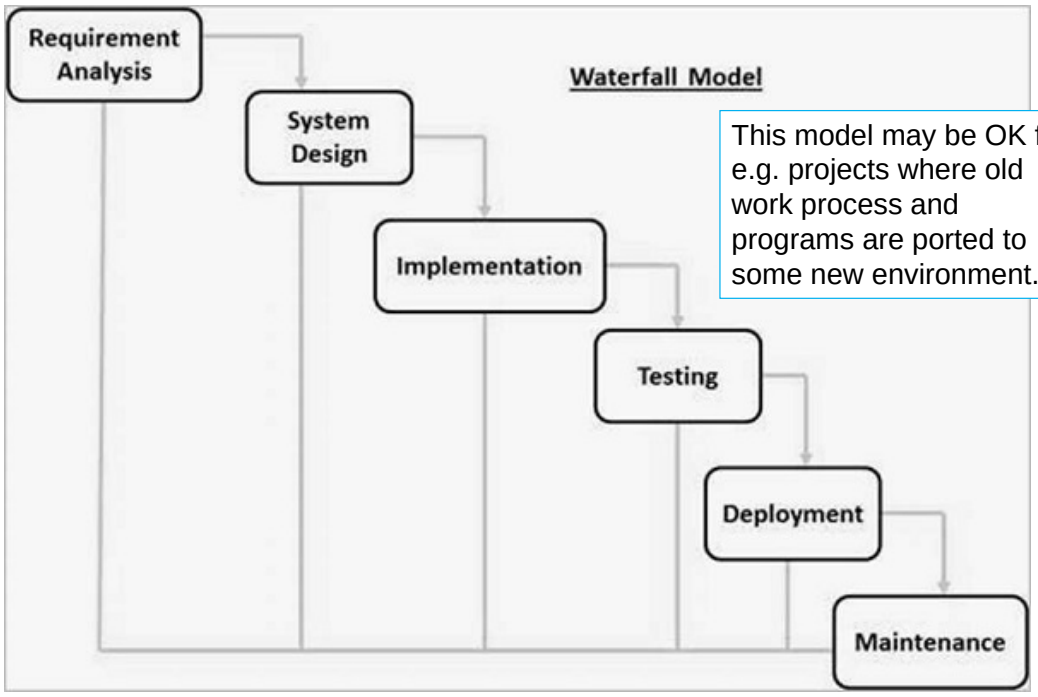
3.3823
software life cycle (SLC)
SLC
1. project-specific sequence of activities that is created by mapping the activities of a standard onto a selected software life cycle model (SLCM) *[IEEE 730-2014 IEEE Standard for Software Quality Assurance Processes,* 3.2*]*
2. software system or software product cycle initiated by a user need or a perceived customer need and terminated by discontinued use of the product or when the software is no longer available for use.

waterfall model (the oldest)

The waterfall model was the first one, it show the separate phases of SDLC.

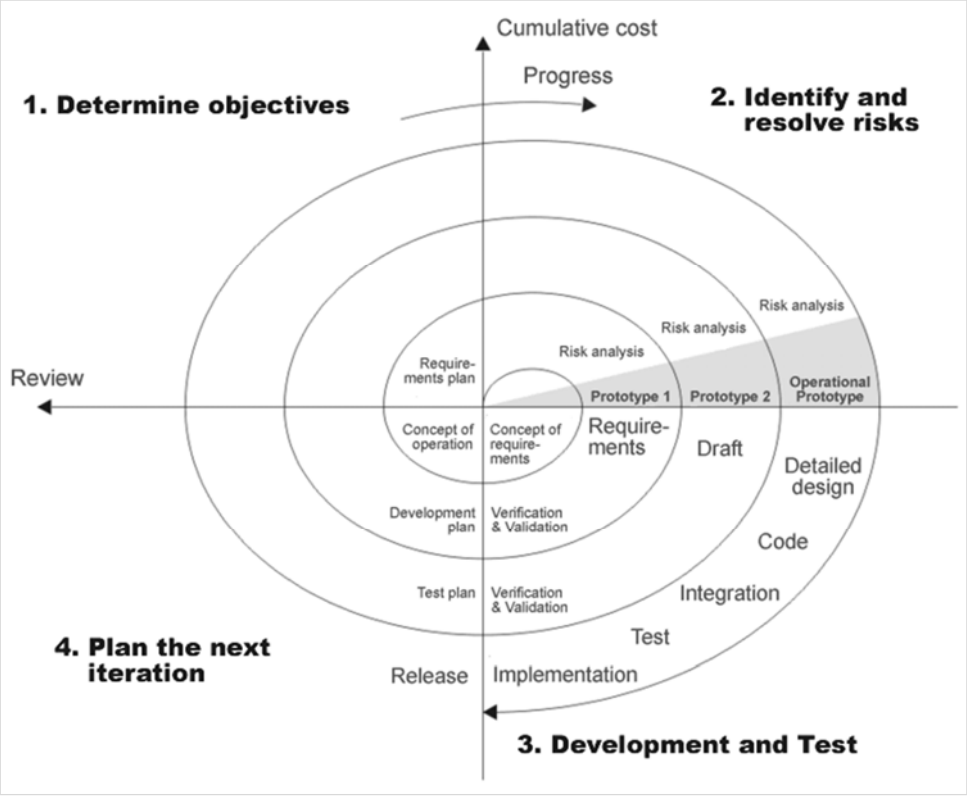
However, it was ”fixed”; after one phase you did not iterate. So you have to do it ”right at the first try”, which usually do not happen.

At the end, after the whole project is ready, customer sees the result (deployment).



This model may be OK for e.g. projects where old work process and programs are ported to some new environment.

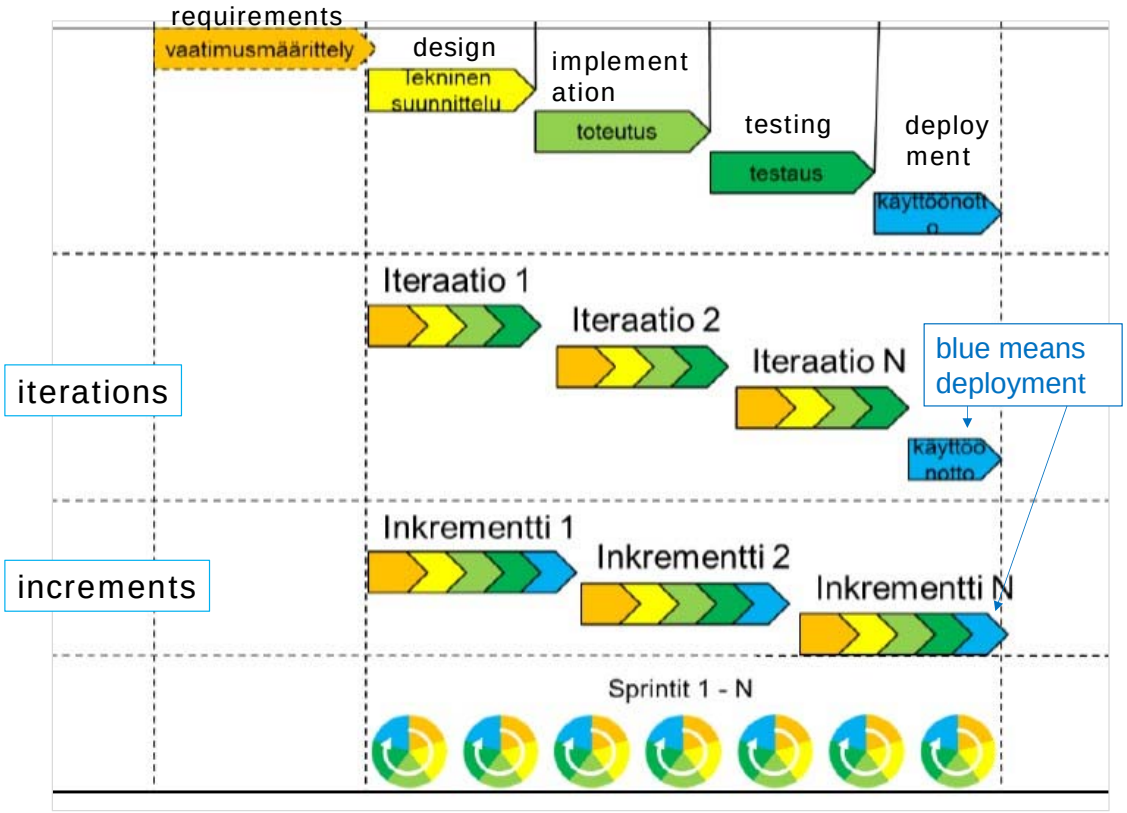
Boehm’s Spiral model (1986)



iterative vs. incremental

At iterative way you make iterations, and at the end you publish (deploy) the product.

At incremental way you make iteration, and deploy a product version after every increment.



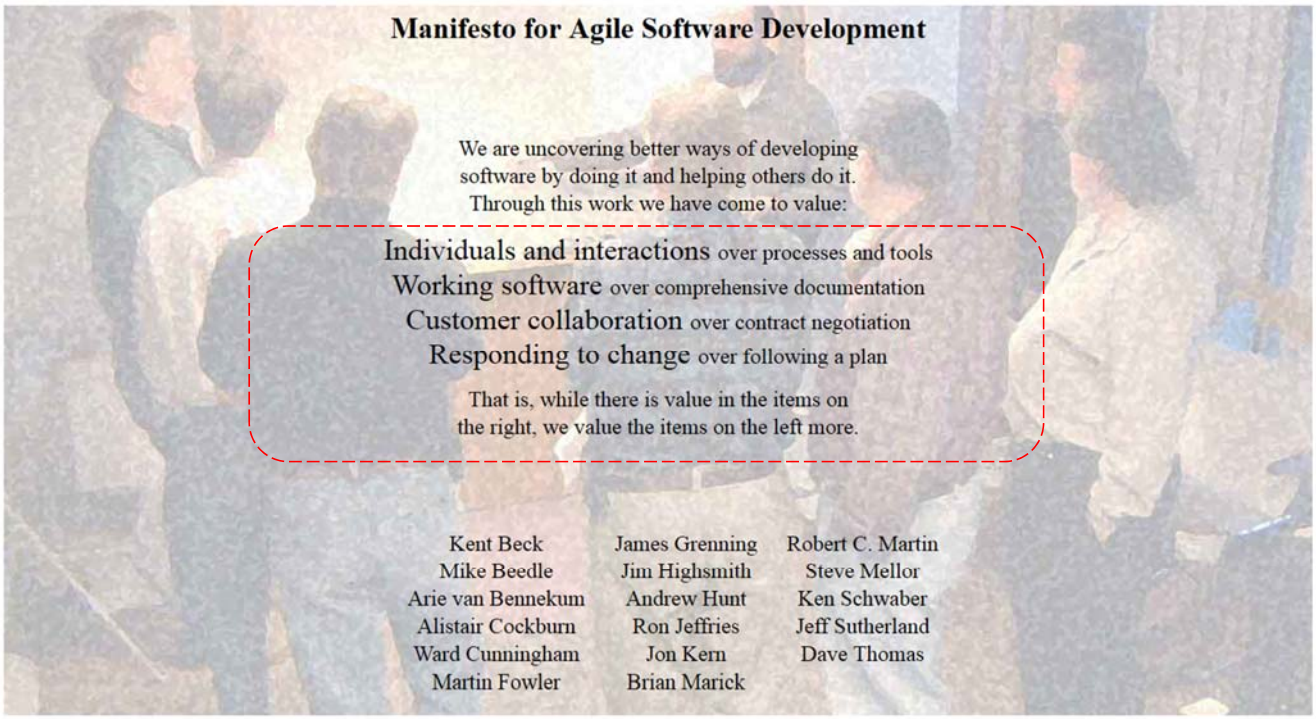
[picture source: Apotti files, in Finnish]

agile

Agile Manifesto

The core is there

But still the small text is also needed, although it is less important.



[https://agilemanifesto.org/]

Principles behind the Agile Manifesto

We follow these (12) principles:

- Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
- Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
- **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must **work together** daily throughout the project.
- Build projects around **motivated** individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- **Simplicity** -- the art of maximizing the amount of work not done -- is essential.
- The best architectures, requirements, and designs emerge from **self-organizing teams**.
- At regular intervals, the team **reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.
-

[https://agilemanifesto.org/principles.html]

Tampereen yliopisto
Tampere University

Well, not
just that…

Tampereen yliopisto
Tampere University

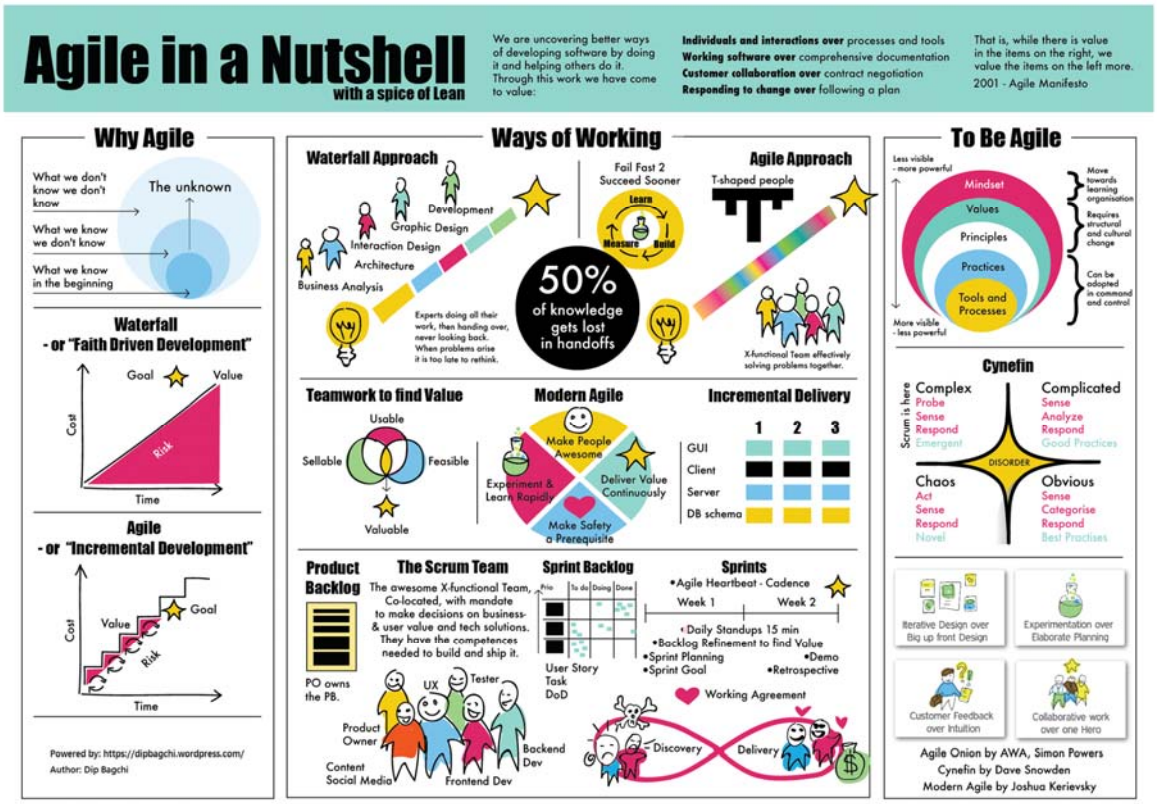
Scrum Master

The scrum master is the team role responsible for ensuring the team lives agile values and principles and **follows the processes and practices** that the team agreed they would use.

The responsibilities of this role include:

- clearing obstacles
- establishing an environment where the team can be effective
- addressing team dynamics
- ensuring a good relationship between the team and product owner as well as others outside the team
- protecting the team from outside interruptions and distractions.

Some good details here



agile, videos and glossary

At YouTube there are many videos e.g. by Mountain Goat Software (**www.mountaingoatsoftware.com**/) or Mike Cohn.

Agile Estimating and Planning: Planning Poker - Mike Cohn
https://www.youtube.com/watch?v=MrIZMuvjTws

You may also check:
https://www.agilealliance.org/agile101/agile-glossary/

planning poker

A playful approach to **estimation**, used by many Agile teams.

The team meets in presence of the customer or Product Owner. Around the table, each team member holds a set of playing cards, bearing numerical values appropriate for points estimation of a user story.

The Product Owner briefly states the intent and value of a story. Each member of the development team silently picks an estimate and readies the corresponding card, face down. When everyone has taken their pick, the cards are turned face up and the estimates are read aloud.

The two (or more) team members who gave **the high and low estimate** justify their reasoning. After brief discussion, the team may seek convergence toward a **consensus estimate** by playing one or more further rounds.

(sometimes you may use this "wrong" and estimate working hours, or days at start)

Pair programming (also documenting etc.)

Pair programming consists of **two programmers sharing a single workstation** (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the "**driver**", the other, also actively involved in the programming task but focusing more on overall direction is the "**navigator**"; it is expected that the programmers swap roles every few minutes or so.

More simply "pairing"; the phrases "paired programming" and "programming in pairs" are also used, less frequently. **In general it is just working in pairs, together.**

Common Pitfalls

• both programmers must be **actively** engaging with the task throughout a paired session, otherwise no benefit can be expected

• a simplistic but often raised objection is that pairing "doubles costs"; that is a misconception based on equating programming with typing – however, one should be aware that this is the worst-case outcome of poorly applied pairing

• at least the driver, and possibly both programmers, are expected to keep up a running commentary; pair programming is also "**programming out loud**" – if the driver is silent, the navigator should intervene

• **pair programming cannot be fruitfully forced** upon people, especially if relationship issues, including the most mundane (such as personal hygiene), are getting in the way; solve these first!

[www.agilealliance.org/glossary/]

refactoring code, ”fine-tuning”

Refactoring consists of **improving the internal structure** of an existing program’s source code, while preserving its external behavior. For example, your “brute force” code is tested and works, but **you re-write it to be a short elegant and well-commented code**.

The following are claimed benefits of refactoring:

- refactoring improves objective attributes of code (length, duplication, coupling and cohesion, cyclomatic complexity) that correlate with ease of maintenance
- refactoring helps code understanding
- refactoring encourages each developer to think about and understand design decisions, in particular in the context of collective ownership / collective code ownership
- refactoring favors the emergence of reusable design elements (such as design patterns) and code modules.

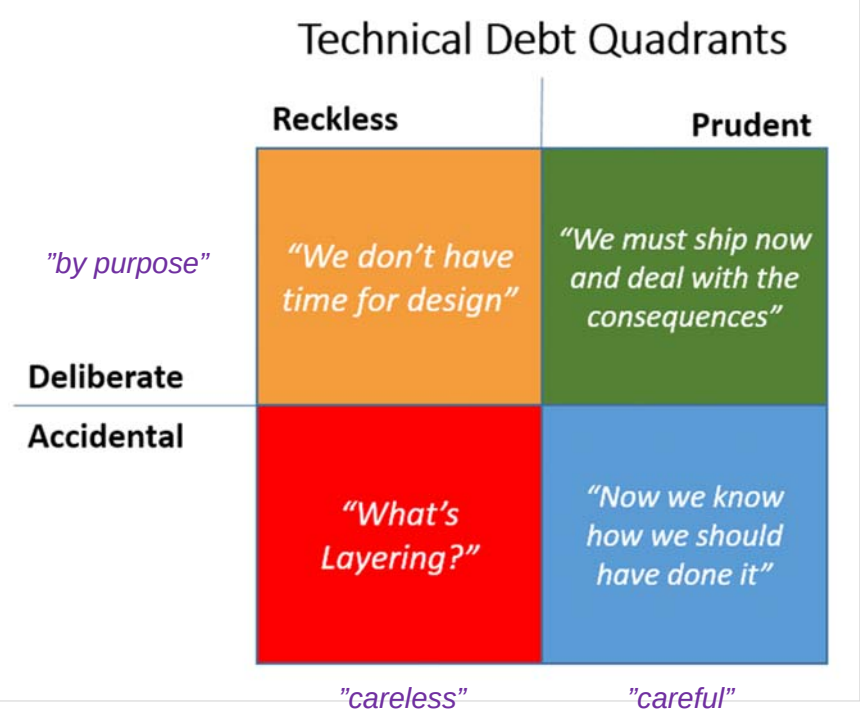
Use e.g. SonarQube tool to find out code quality.

That code needs refactoring, to lower technical debt.

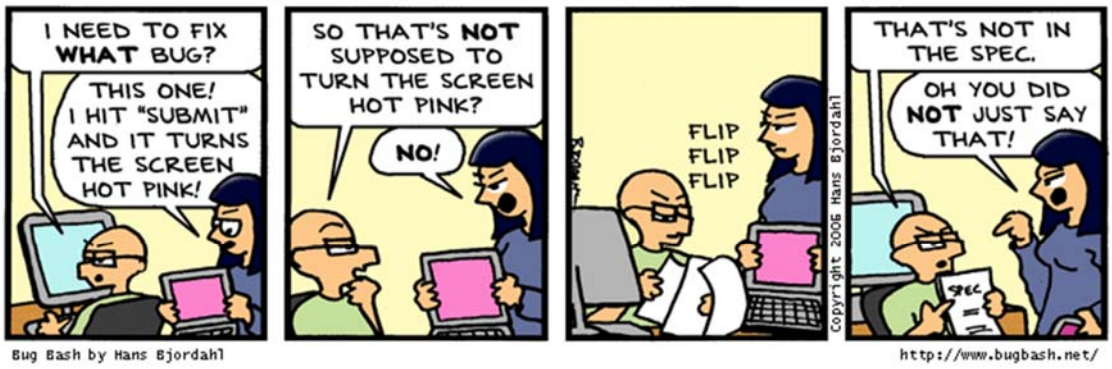
technical debt

Technical debt is a metaphor for all of the **shortcuts, hacks, and poor design choices** made for a given software system that compromised its quality, usually in an attempt to meet a deadline.

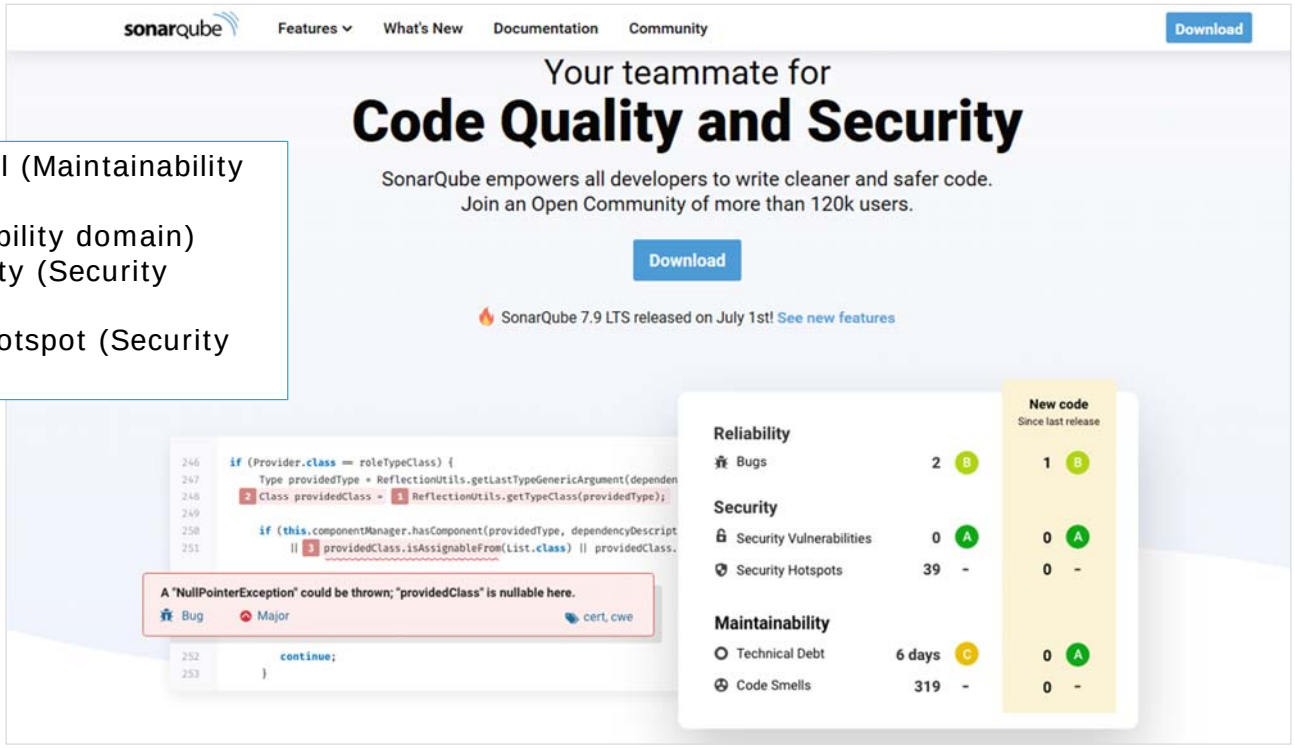
It can be an appropriate business decision to take on technical debt, but if such debt is allowed to grow, the lack of quality of the system may eventually make it too expensive to maintain, at which point the business or team may need to declare “technical bankruptcy” and rewrite the application rather than continue to try to maintain it.

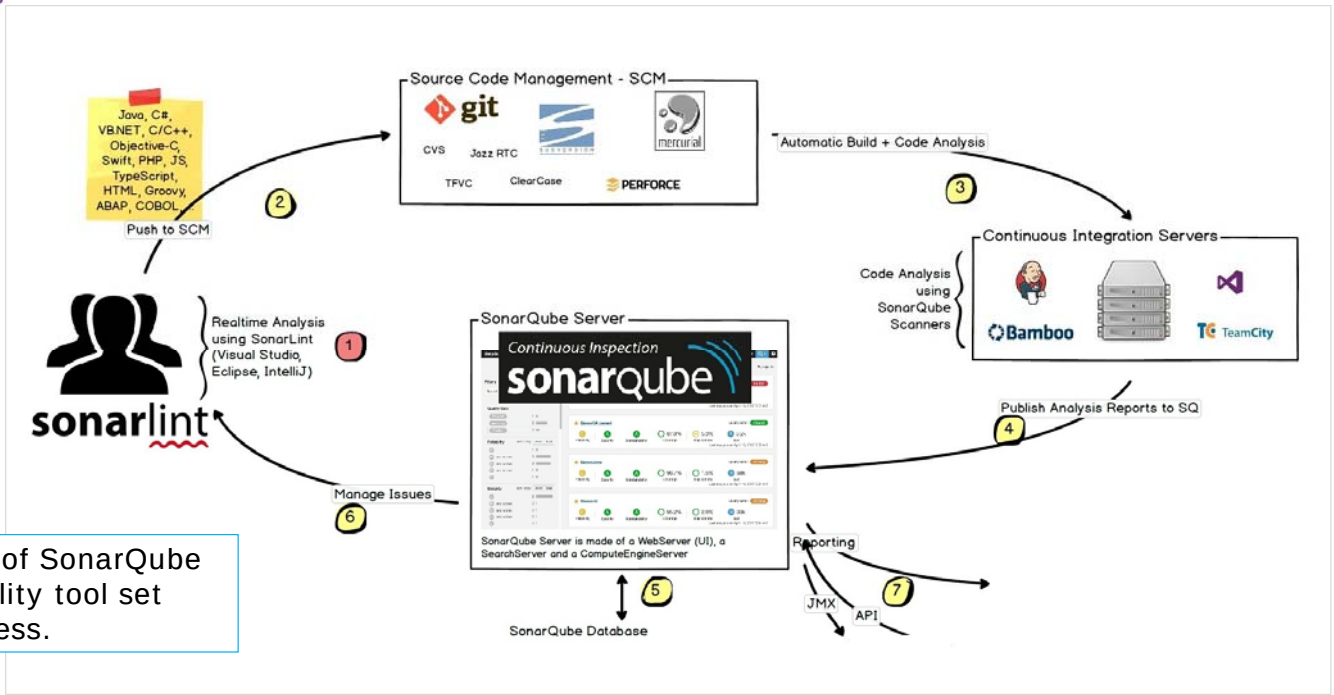


[https://deviq.com/technical-debt/]



- Code Smell (Maintainability domain)
- Bug (Reliability domain)
- Vulnerability (Security domain)
- Security Hotspot (Security domain)





Example of SonarQube code quality tool set and process.

Velocity

At the end of each iteration, the team adds up effort estimates associated with user stories that were **completed** during that iteration. This total is called **velocity**.

Knowing velocity, the team can compute (or revise) an estimate of how long the project will take to complete, based on the estimates associated with remaining user stories and assuming that velocity over the remaining iterations will remain approximately the same. This is generally an accurate prediction, even though rarely a precise one.

"Worked example:" an agile team has started work on an iteration, planning to complete stories A and B, estimated at 2 points each, and story C, estimated at 3 points. At the end of the iteration, stories A and B are 100% complete but C is only 80% complete.

Agile teams generally acknowledge only two levels of completion, **0% done or 100% done**. Therefore, C is not counted toward velocity, and velocity as of that iteration is 4 points.

Suppose the user stories remaining represent a total of 40 points; the team's forecast of the remaining effort for the project is then 10 iterations.

Usually the team's velocity will grow along the project.

Definition of done (DoD)

The team agrees on, and displays prominently somewhere in the team room, a list of **criteria which must be met before a product increment "often a user story" is considered "done"**. Failure to meet these criteria at the end of a sprint normally implies that the work should not be counted toward that sprint's velocity.

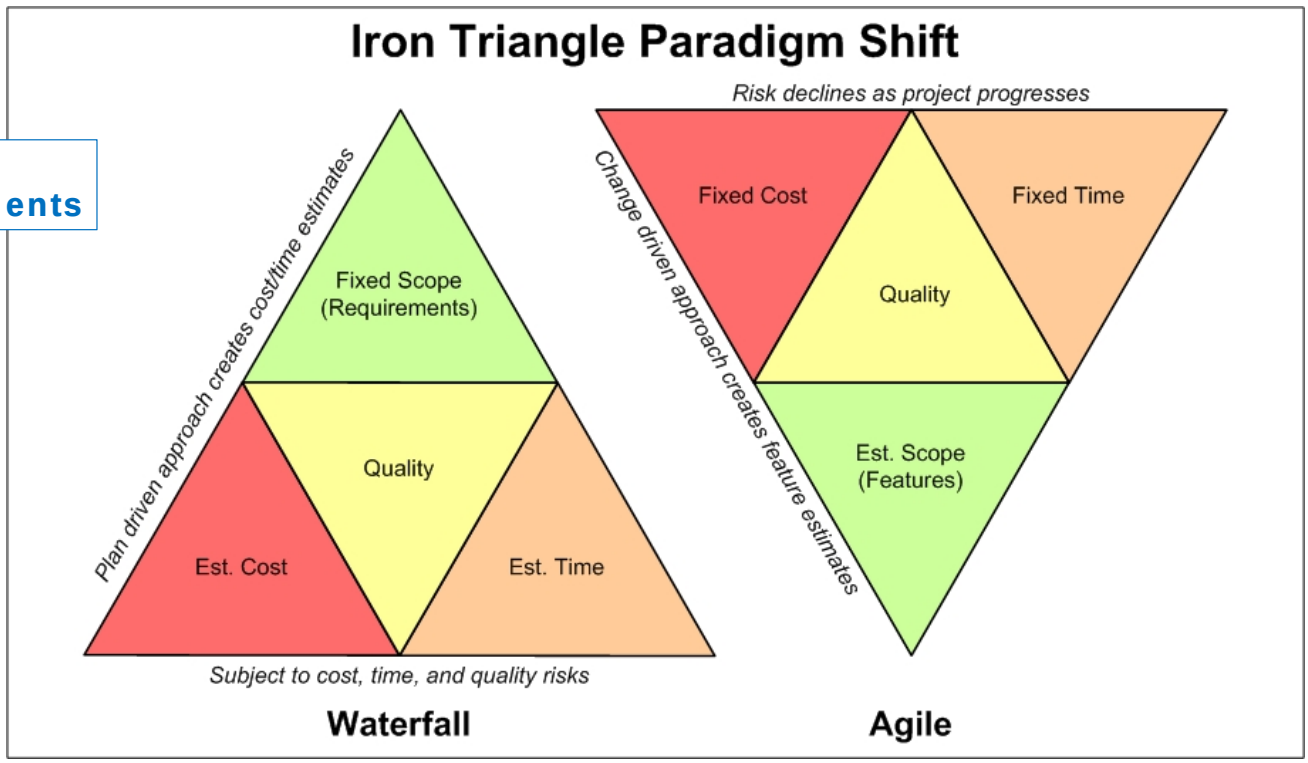
Some teams use the term "Done List" or "Done Checklist".

- the Definition of Done limits the cost of rework once a feature has been accepted as "done"
- having an explicit contract **limits the risk of misunderstanding** and conflict between the development team and the customer or product owner.

DoD for a feature may be e.g. peer reviewed, unit tested (CI pipeline), checked against requirements, or merge/pull request (some other developer than author pushes code to version control).

Tampereen yliopisto
Tampere University

Agile enhancements



[http://iq3group.blogspot.com/2013/01/decision-analysis-dcf-is-waterfall-rom.html]

Tampereen yliopisto
Tampere University

XP = Extreme Programming

Extreme Programming (**XP**) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

The general characteristics where XP is appropriate were described by Don Wells on ***www.extremeprogramming.org***:

- Dynamically changing software requirements
- Risks caused by fixed time projects using new technology
- Small, co-located extended development team
- The technology you are using allows for automated unit and functional tests.

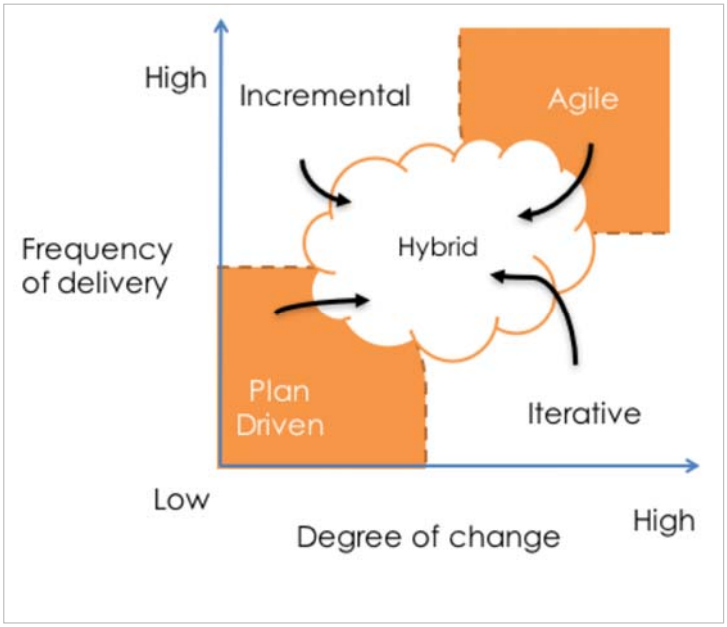
Due to XP's specificity when it comes to it's full set of software engineering practices, there are several situations where you may not want to fully practice XP. The post When is XP Not Appropriate on the C2 Wiki is probably a good place to start to find examples where you may not want to use XP.
While you can't use the entire XP framework in many situations, that shouldn't stop you from **using as many of the practices as possible given your context**.

[www.agilealliance.org/glossary/]

hybrid agile models

Blended Agile is the combination of two or more established Agile methods, techniques, or frameworks.

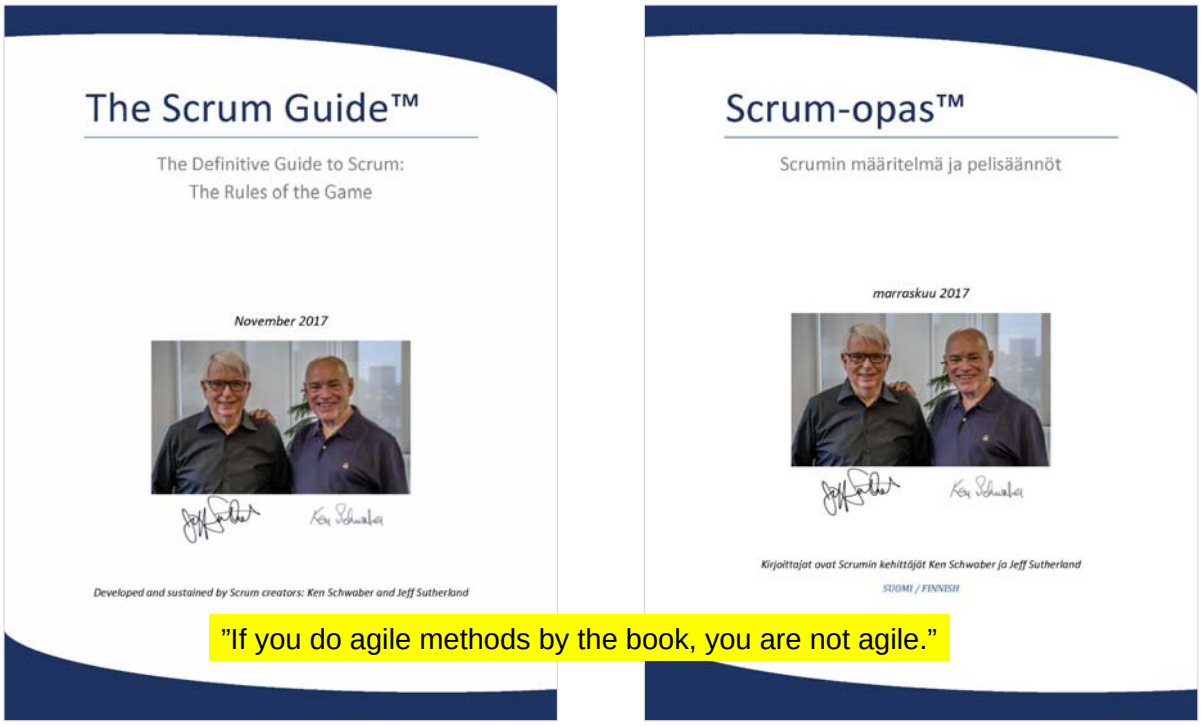
Hybrid Agile is the combination of Agile methods with other non-Agile techniques.



[https://www.agilealliance.org/what-is-hybrid-agile-anyway/]

Scrum

www.scrumguides.org



"If you do agile methods by the book, you are not agile."

Scrum values [Scrum Guide 2017]

When the values of **commitment**, **courage**, **focus**, **openness** and **respect** are embodied and lived by the Scrum Team, the Scrum pillars of **transparency**, **inspection**, and **adaptation** come to life and build trust for everyone.

(FI: Scrumin arvot ovat **sitoutuminen**, **rohkeus**, **keskittyminen**, **avoimuus** ja **kunnioitus**. Arvojen mukaan toimiessaan Scrum-tiimi vahvistaa Scrumin kolmea tukijalkaa: **läpinäkyvyyttä**, **tarkastelua** ja **sopeuttamista**. Tämä puolestaan kasvattaa luottamusta.)

Scrum, basic terms

Product Backlog (PB) (FI: kehitysjono, kehityspino, tuotepino)

Sprint (FI: pyrähdys), iteration

Sprint Backlog (SB)

Work item

Task (FI: tehtävä)

Planning poker

Velocity (FI: vauhti, kehityskyvykkyys)

Scrum Master (FI: Scrummaster)

Product Owner (FI: tuoteomistaja)

Refactoring (FI: refaktorointi, kodin hiominen, koodin parantelu)

Burndown Chart (or Burnup Chart) (FI: edistymiskäyrä, edistymisen seuranta)

Definition of Done (DoD) (FI: valmiin määritelmä)

Scrum team [Scrum Guide 2017]

The Scrum Team consists of a **Product Owner**, the **Development Team**, and a **Scrum Master**. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.

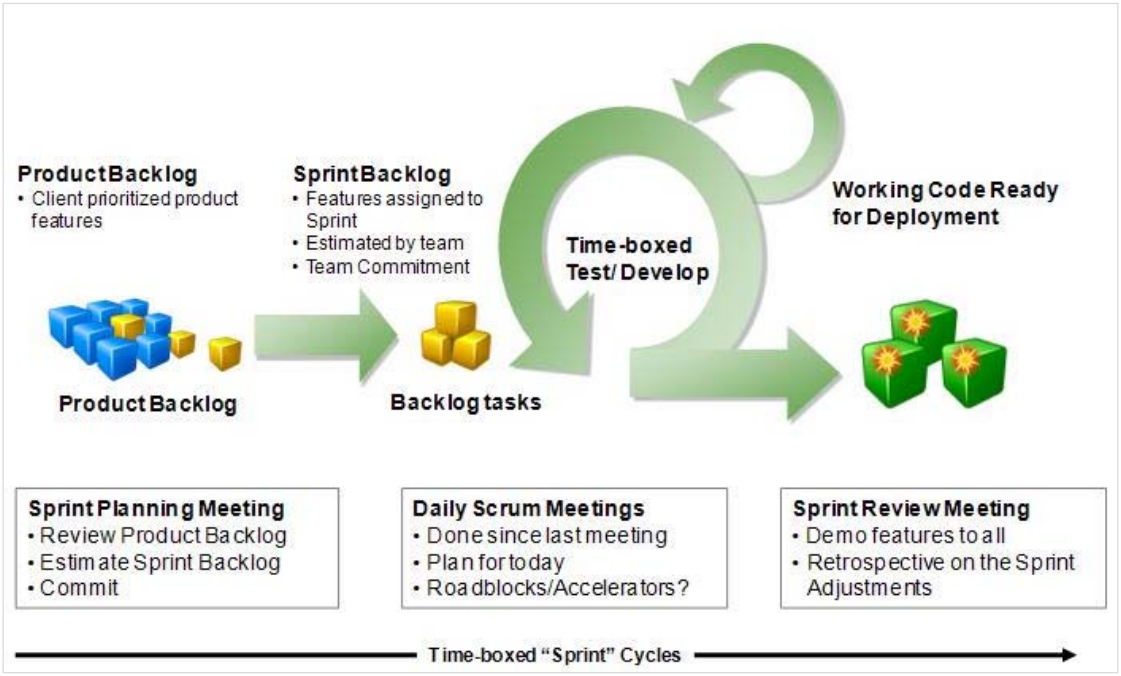
Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of “Done” product ensure a potentially useful version of working product is always available.

- - -

In Finland, often Scrum Master means, or does the same work than, a Project Manager. Which is actually against Scrum Guide.

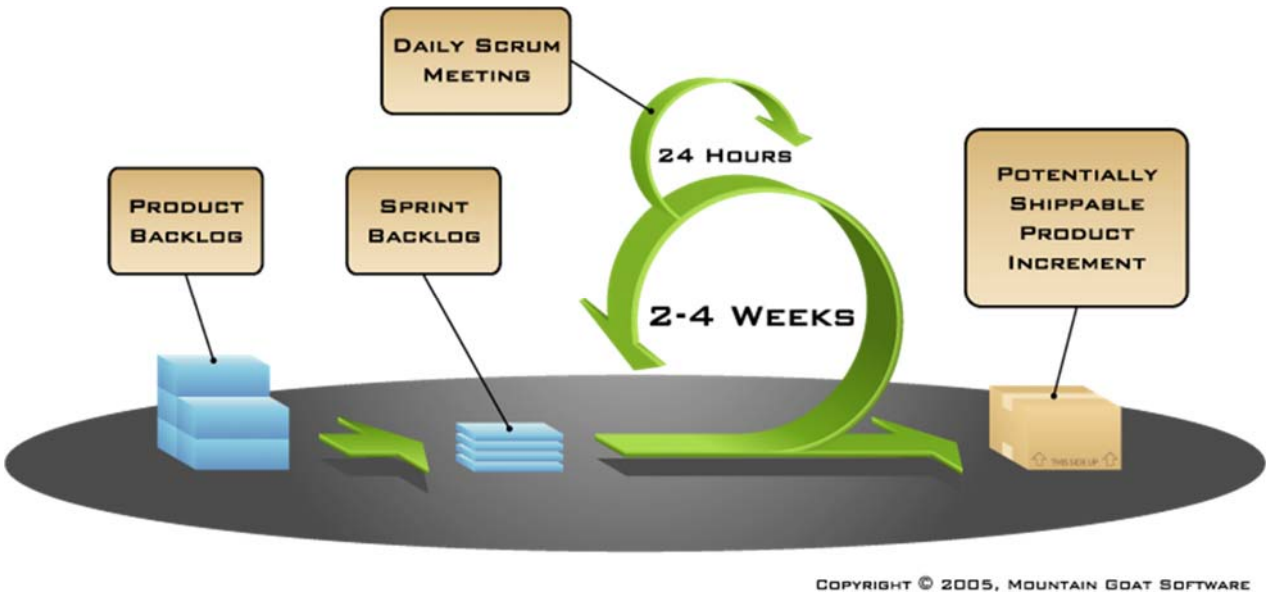
Scrum team is supposed to be an experienced TEAM.

Scrum basics



Sprint retrospective meeting

Putting it all together



[www.mountaingoatsoftware.com/scrum]

Tampereen yliopisto
Tampere University

Scrum recommendations

- At the **end of every working day, there should be a demoable product version available**. If you don't get your work item finished (tested to work) during a working day, you do not merge it to version control, you continue with it on next working day.
- **No overtime working** on evenings or weekends.
- Ideally, team should have **only one project at a time** to work with.
- **Team is supposed to be experienced**, everybody knows each other (strenghts and weaknesses).

Tampereen yliopisto
Tampere University

Scrum

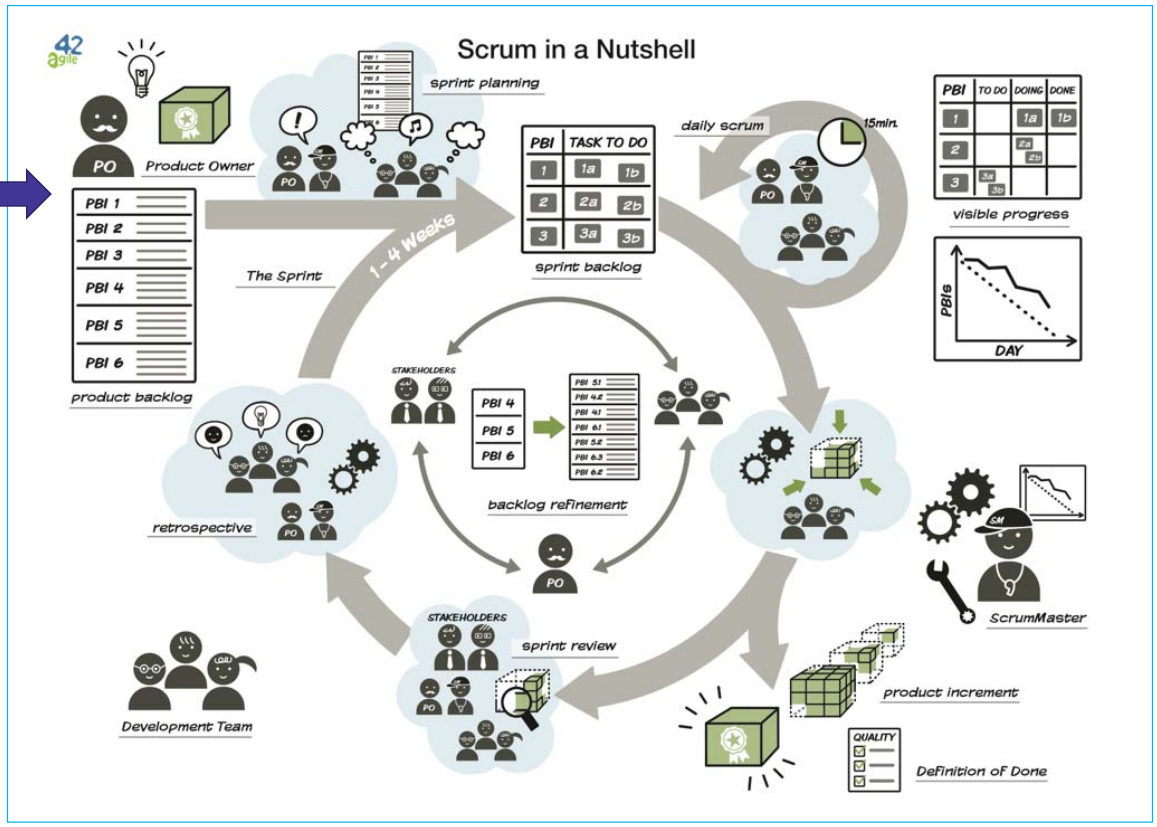
Definition of "Done"

When a Product Backlog item or an Increment is described as "Done", **everyone must understand what "Done" means**. Although this may vary significantly per Scrum Team, members must have a shared understanding of what it means for work to be complete, to ensure transparency. This is the definition of "Done" for the Scrum Team and is used to assess when work is complete on the product Increment.

As Scrum Teams mature, it is expected that their definitions of "Done" will expand to include more stringent criteria for higher quality. New definitions, as used, may uncover work to be done in previously "Done" increments. Any one product or system should have a definition of "Done" that is a standard for any work done on it.

[Scrum Guide 2017]

Scrum

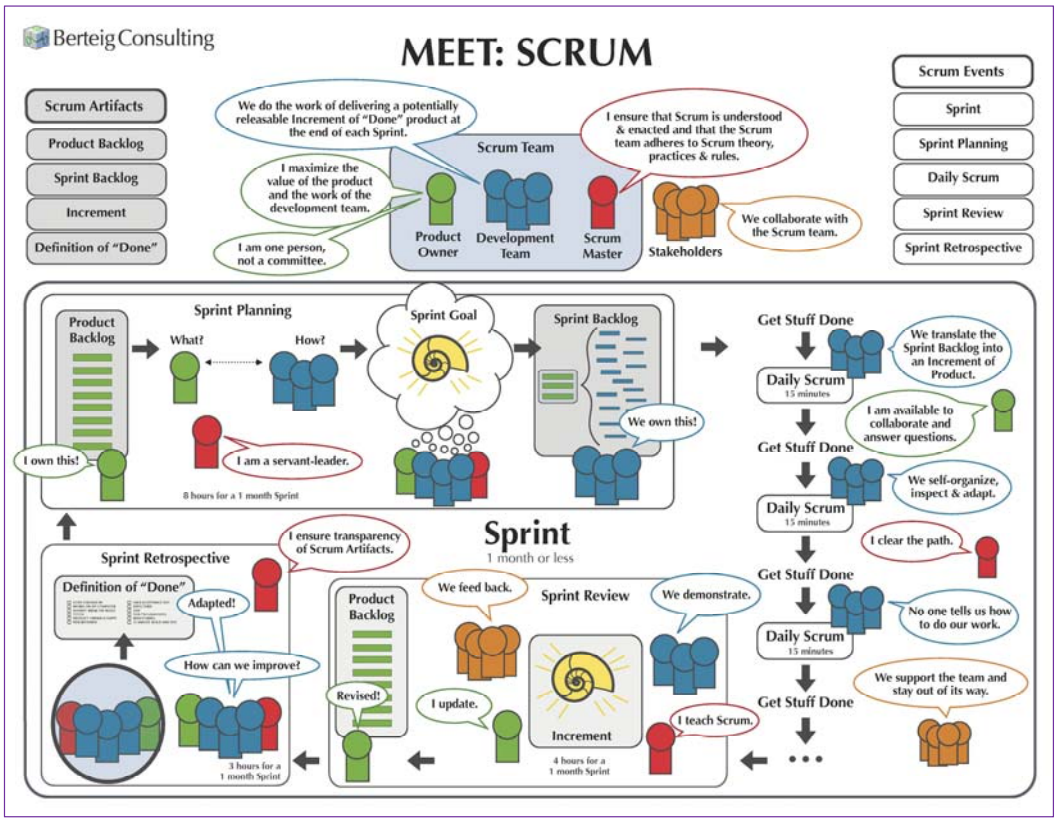


Sprint planning

Sprint work

Sprint review

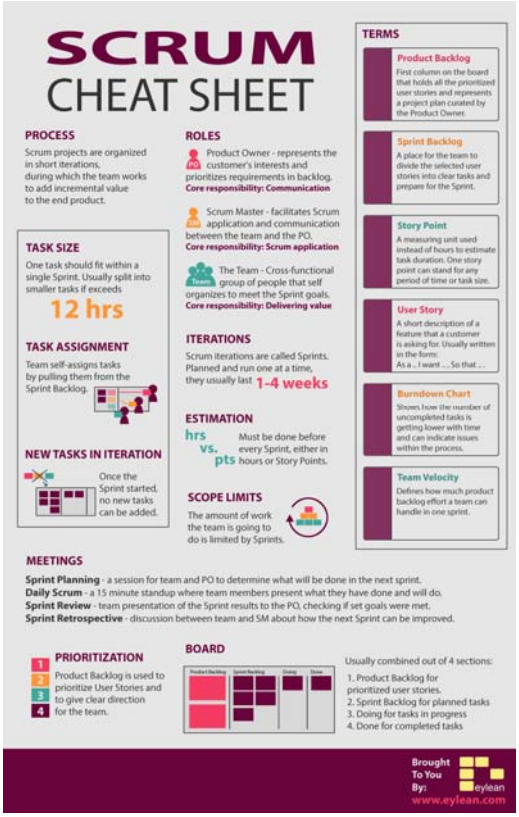
Sprint retrospective

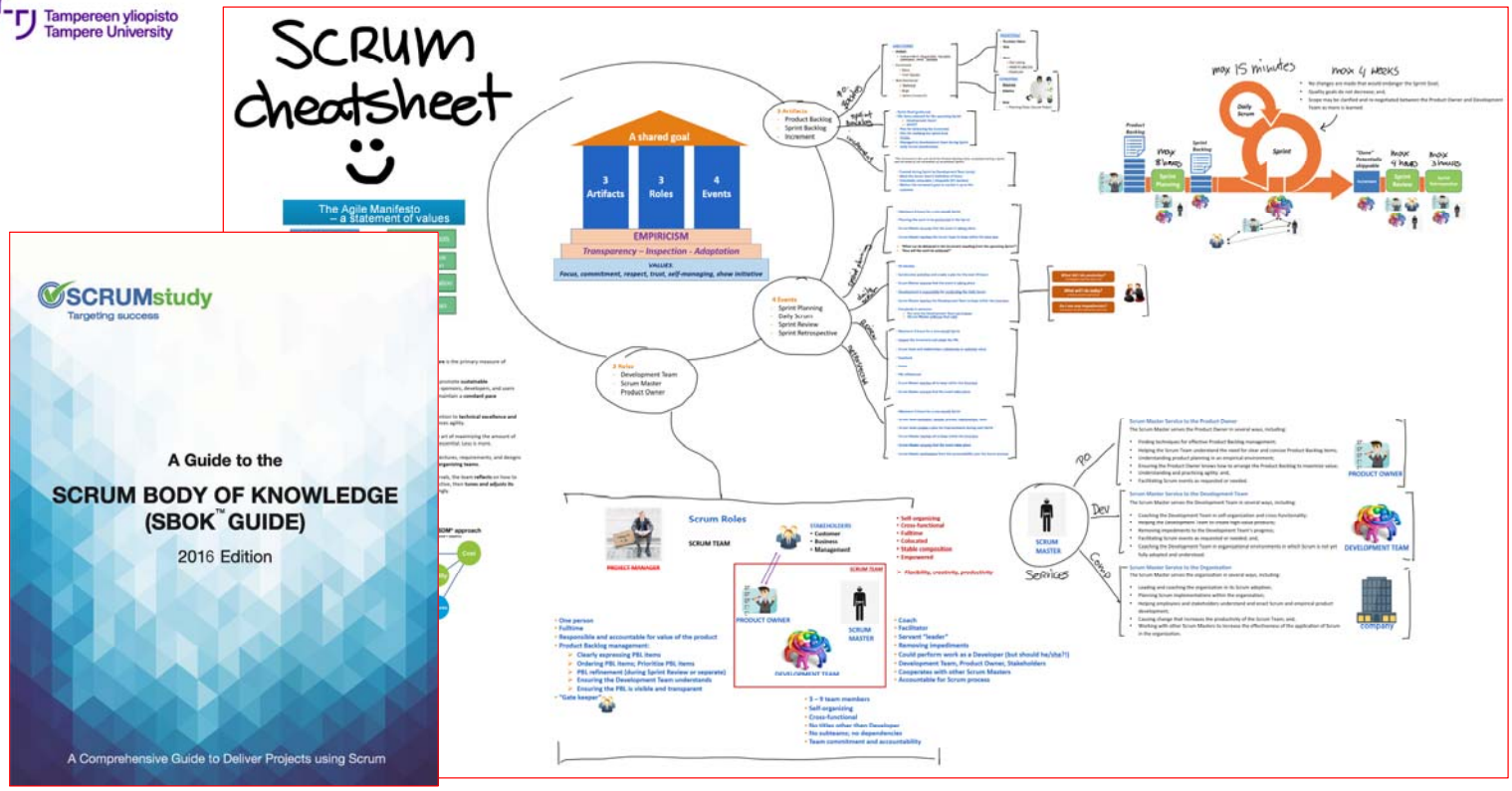


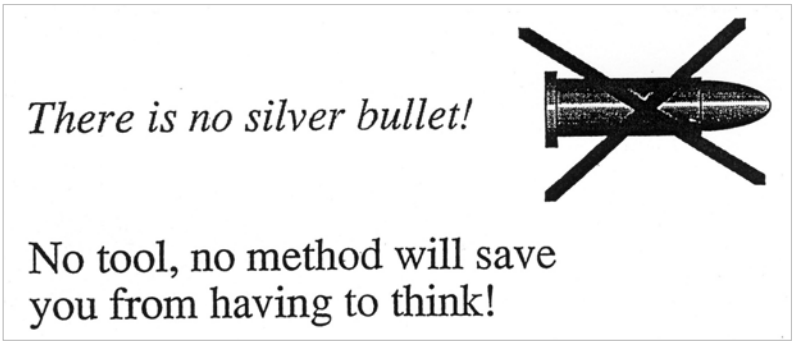
There are many agile and Scrum videos at YouTube, for example

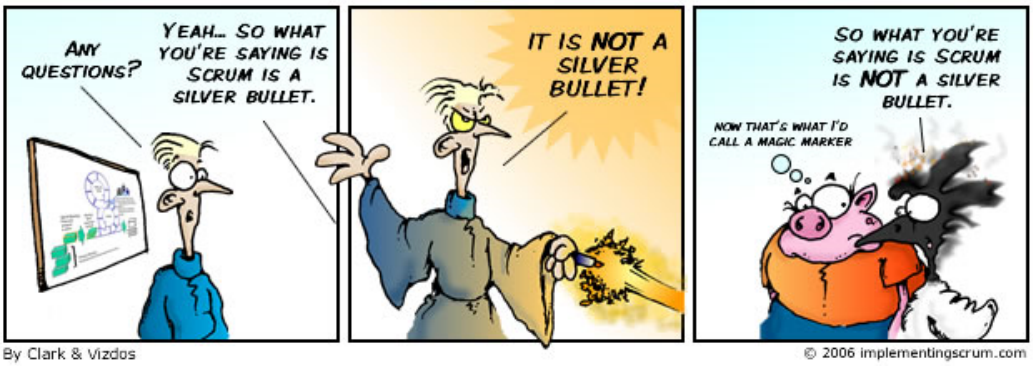
Introduction to Scrum - 7 Minutes
https://www.youtube.com/watch?v=9TycLR0TqFA

Intro to Scrum in Under 10 Minutes
https://www.youtube.com/watch?v=XU0llRltyFM




SCRUM
cheatsheet
The Agile Manifesto
– a statement of values
A shared goal
3
Artifacts
3
Roles
4
Events
EMPIRICISM
Transparency – Inspection - Adaptation
Product Backlog
Sprint Backlog
Increment
Sprint Planning
Daily Scrum
Sprint Review
Sprint Retrospective
Development Team
Scrum Master
Product Owner
max 15 minutes
max 4 weeks
Sprint
SCRUMstudy
Targeting success
A Guide to the
SCRUM BODY OF KNOWLEDGE
(SBOK™ GUIDE)
2016 Edition
A Comprehensive Guide to Deliver Projects using Scrum
Scrum Roles
SCRUM TEAM
PROJECT MANAGER
PRODUCT OWNER
SCRUM MASTER
DEVELOPMENT TEAM
STAKEHOLDERS
Customer
Business
Management
PO
Dev
Comp
Services
PRODUCT OWNER
DEVELOPMENT TEAM
company


There is no silver bullet!
No tool, no method will save
you from having to think!


ANY QUESTIONS?
YEAH... SO WHAT YOU'RE SAYING IS SCRUM IS A SILVER BULLET.
IT IS NOT A SILVER BULLET!
SO WHAT YOU'RE SAYING IS SCRUM IS NOT A SILVER BULLET.
NOW THAT'S WHAT I'D CALL A MAGIC MARKER
By Clark & Vizdos
© 2006 implementingscrum.com

Burndown / burnup chart

The team displays, somewhere on a wall of the project room, a large graph relating the **quantity of work remaining** (on the vertical axis) and the time elapsed since the start of the project (on the horizontal, showing future as well as past). This constitutes an "information radiator", provided it is updated regularly. Two variants exist, depending on whether the amount graphed is for the work remaining in the iteration ("sprint burndown") or more commonly the entire project ("product burndown").

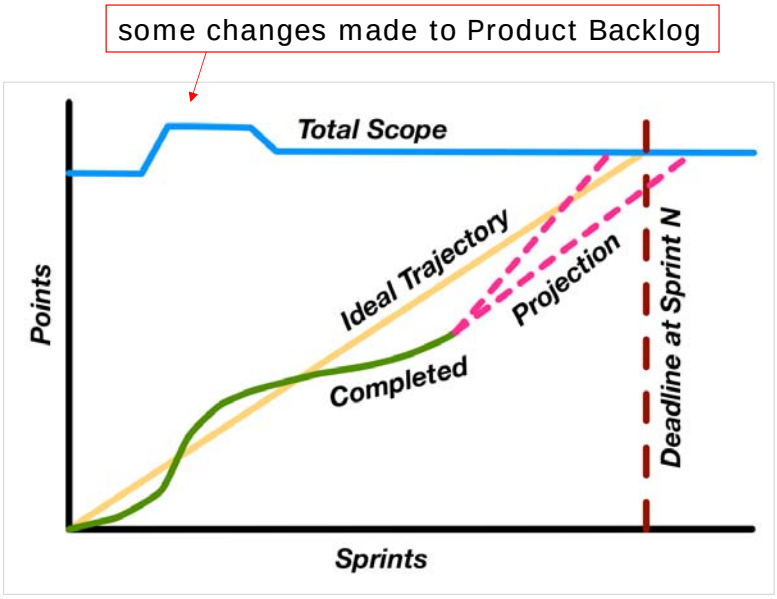
The term "burn chart" is sometimes encountered, possibly as a generalization covering variants such as the "burn up chart".

This practice results in **up-to-date project status** being not only visible, but in fact shoved into the faces of everyone involved: as a result it **encourages** the team to confront any difficulties sooner and more decisively.

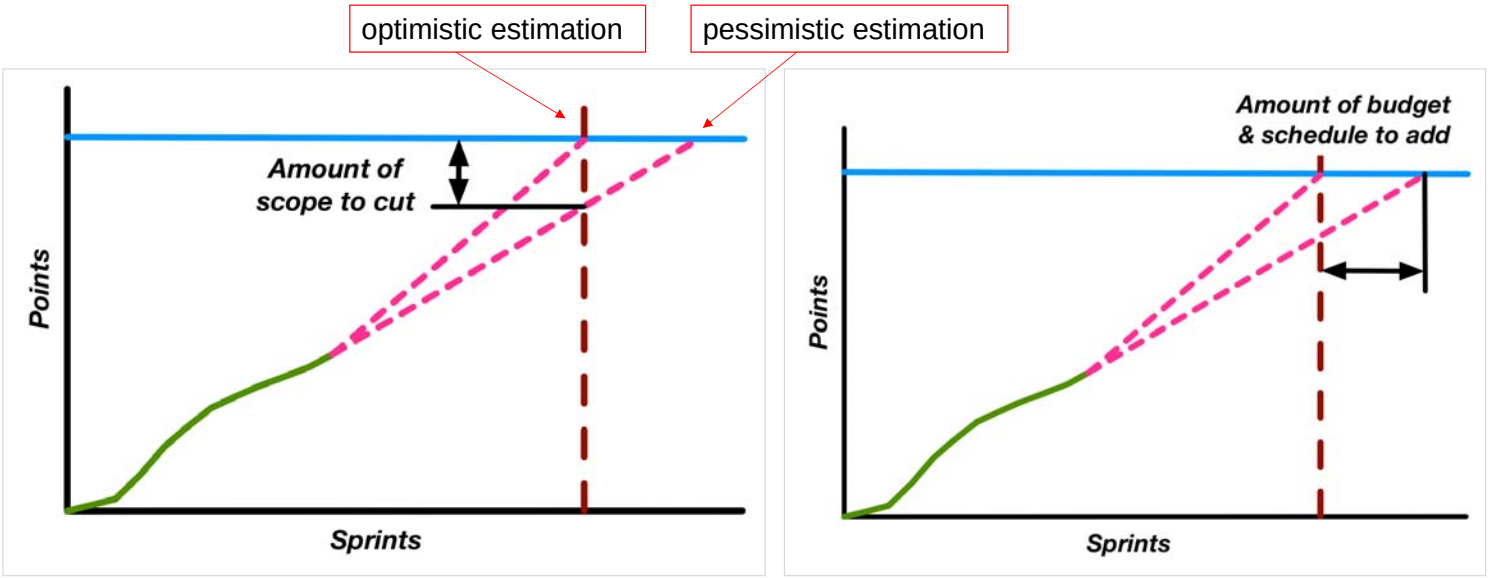
The original chart was **Burn-Up**, but you may also use **Burn-Down** chart, to show how little work is left, still to be done.

Burnup chart example, 1

- Horizontal (X) Axis – Amount of **Time** (in sprints)
- Vertical (Y) Axis – Amount of **Work** to be Done (in points)
- Blue Line – Total points; the overall **scope** of the project over time.
- Green Line – **Completed** points; the team's progress towards their goal over time.
- Brown Dotted Line – A budget or calendar **deadline**.
- Yellow Line – The **ideal** pace needed to meet the deadline.
- Pink Dotted Line – A high and low projection of likely future progress, based on historical velocity data. (Optional)



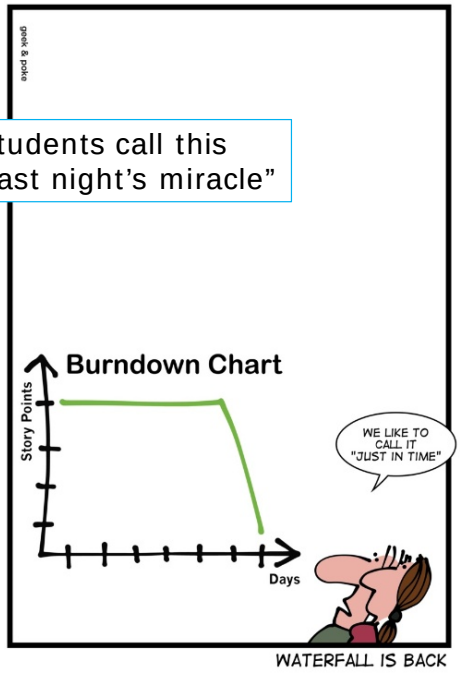
Burnup chart example, 2

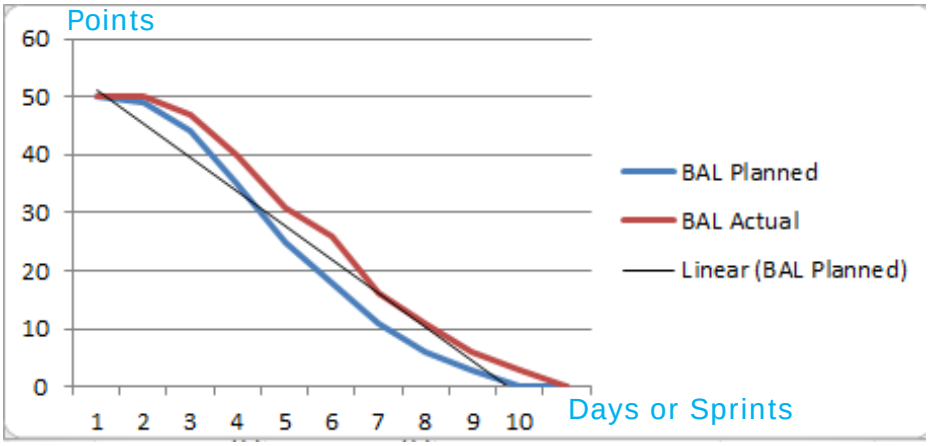


At the end of Scrum projects, there are usually some (less important) work items left in the PB.

Burndown chart, 1

Students call this ”last night’s miracle”





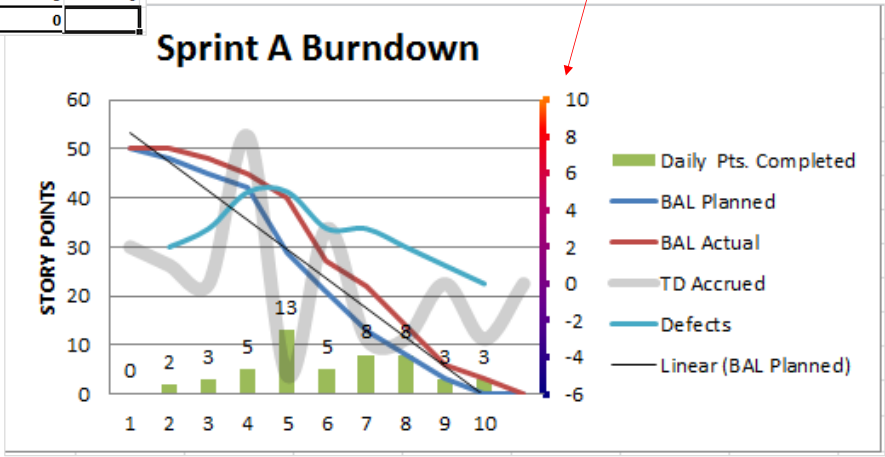
[https://qualitasrebus.com/]

Burndown chart, 2

in table and chart format

Day	BD Planned	BD Actual	BAL Planned	BAL Actual	Daily Pts. Completed	TD Accrued	Defects
			50	50			
1	2	0	48	50	0	2	
2	3	2	45	48	2	1	2
3	3	3	42	45	3	0	3
4	13	5	29	40	5	8	5
5	8	13	21	27	13	-5	5
6	8	5	13	22	5	3	3
7	5	8	8	14	8	-3	3
8	5	8	3	6	8	-3	2
9	3	3	0	3	3	0	1
10	0	3	0	0	3	-3	0
	50	50				0	

TD count

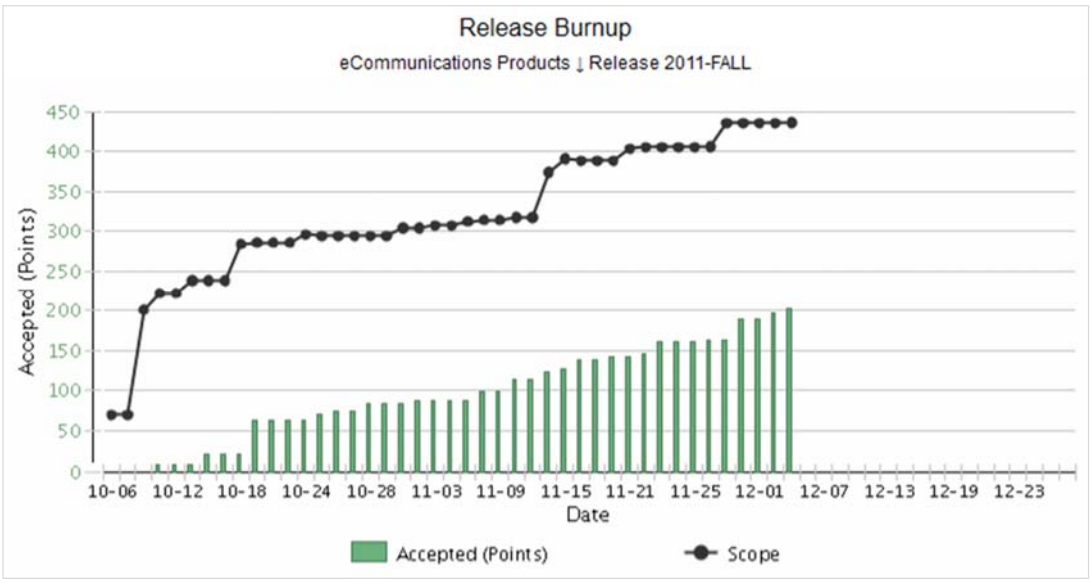


BAL = burndown all left
BD = burndown (daily points)
TD = technical debt

[https://qualitasrebus.com/]

Just because you are Agile, does not mean you eliminated scope creep.

We focus on the features that are highest in priority and deliver early and often to create a ***minimal marketable product (MMP)*** giving us a serious advantage over other methodologies that struggle to understand the WHOLE PROJECT from the very beginning.



[myagilemind.wordpress.com/2011/12/05/scope-creep-agile-is-not-immune-to-it/]

Lean seven principles

”concentrate to the essentials”

[https://hangoutagile.com/wp-content/uploads/2018/09/Lean-Software-Development.jpg]


#93

Get More Refcardz! Visit refcardz.com

DZone Refcardz

brought to you by... VERSIONONE Simplifying Software Delivery

CONTENTS INCLUDE:

- About Lean Software Development
- Getting Started
- Zero Practices
- Daily Standup
- Automated Testing
- Continuous Integration and more...

Getting Started with

Lean Software Development

By Curt Hibbs, Steve Jewett, and Mike Sullivan

ABOUT LEAN SOFTWARE DEVELOPMENT

Lean Software Development is an outgrowth of the larger Lean movement that includes areas such as manufacturing, supply chain management, product development, and back-office operations. Its goal is the same: deliver value to the customer more quickly by eliminating waste and improving quality. Though software development differs from the manufacturing context in which Lean was born, it draws on many of the same principles.

Seven Principles of Lean Software Development

Lean Software Development embodies seven principles, originally described in the book Implementing Lean Software Development: From Concept to Cash[1], by Mary and Tom Poppendieck. Each of these seven principles contributes to the "leaning out" of a software development process.

to accomplish a task, recognizing their efforts, and standing by them when those efforts are unsuccessful.

Optimize the Whole

Optimizing the whole development process generates better results than optimizing local processes in isolation, which is usually done the expense of other local processes.

Lean vs. Agile

Comparing Lean and Agile software development reveals they share many characteristics, including the quick delivery of value to the customer, but they differ in two significant ways: scope and focus. The narrow scope of Agile addresses the development of software and focuses on adaptability to deliver quickly. Lean looks at the bigger picture, the context in which development occurs, and delivers value quickly by focusing on the elimination of waste. As it turns out, they are complementary, and real world processes often draw from both.

Lean

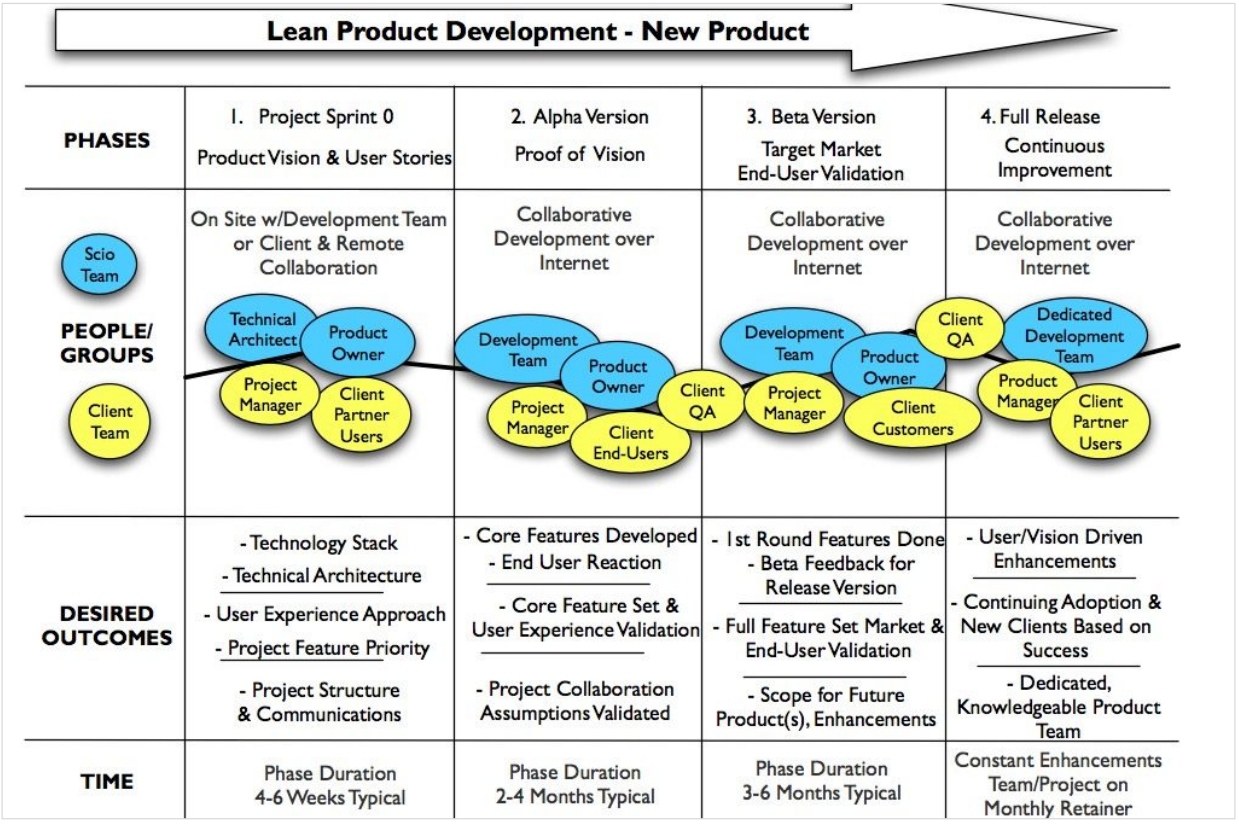
Reason for waste

Eliminating waste

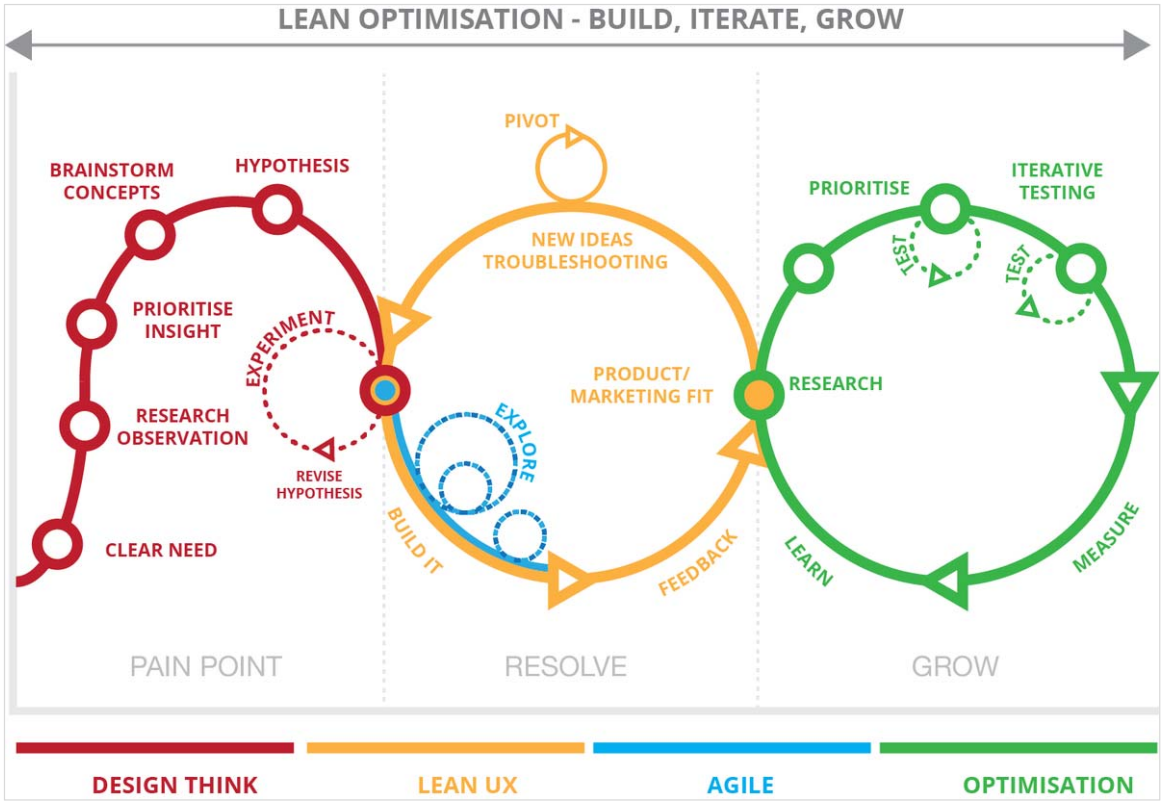
[Agile Lean Development for Waste Management: A Review, 2017]

One example

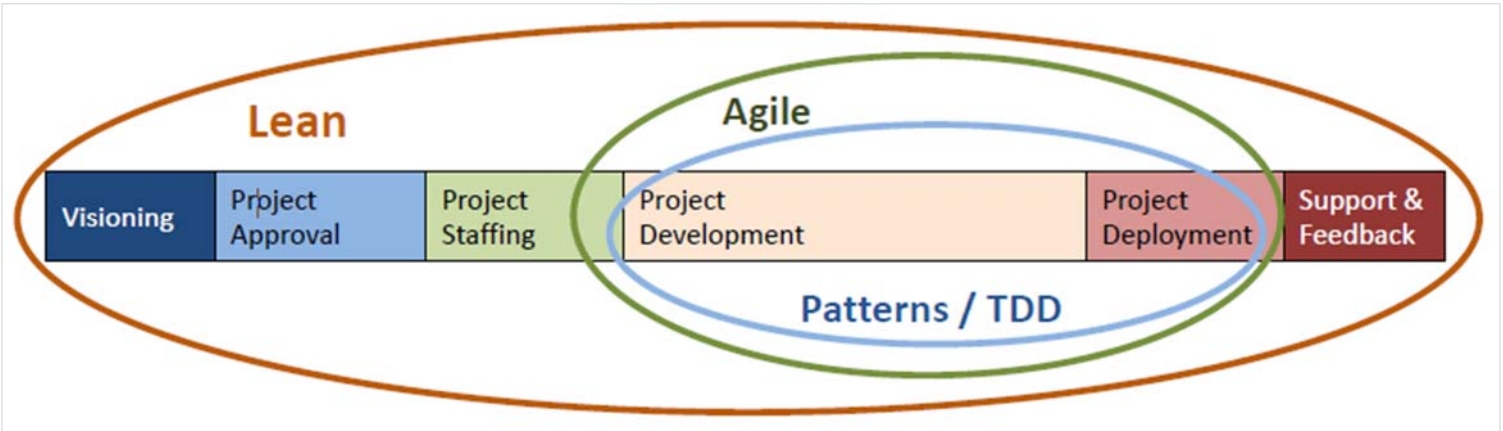
Scio = consulting company



[https://sciodev.com/blog/lean-software-product-development/]



[Craig Sullivan, 2015]



TDD = Test Driven Development

[https://www.infoq.com/news/2008/11/Lean-Agile-Alan-Shalloway/]

MVP = minimum viable product

A common misunderstanding is that MVP is ”the lousiest product which developer dares to send to customer”.

The MVP is that version of **the product that enables a full turn of the Build-Measure-Learn loop with a minimum amount of effort and the least amount of development time**. The minimum viable product lacks many features that may prove essential later on.

However, in some ways, creating a MVP requires extra work: we must be able to measure its impact. For example, it is inadequate to build a prototype that is evaluated solely for internal quality by engineers and designers. We also need to get it **in front of potential customers** to gauge their reactions. We may even need to try selling them the prototype, as we'll soon see.

A minimum viable product (MVP) helps entrepreneurs start the process of learning as quickly as possible. It is not necessarily the smallest product imaginable, though; it is simply the fastest way to get through the Build-Measure-Learn feedback loop with the minimum amount of effort.

Contrary to traditional product development, which usually involves a long, thoughtful incubation period and strives for product perfection, **the goal of the MVP is to begin the process of learning**, not end it. Unlike a prototype or concept test, an MVP is designed not just to answer product design or technical questions. Its goal is to test fundamental business hypotheses.

[Eric Ries: Lean startup, 2011]

MVP = minimum viable product

A common misunderstanding is that MVP is ”the lousiest product which developer dares to send to customer”.

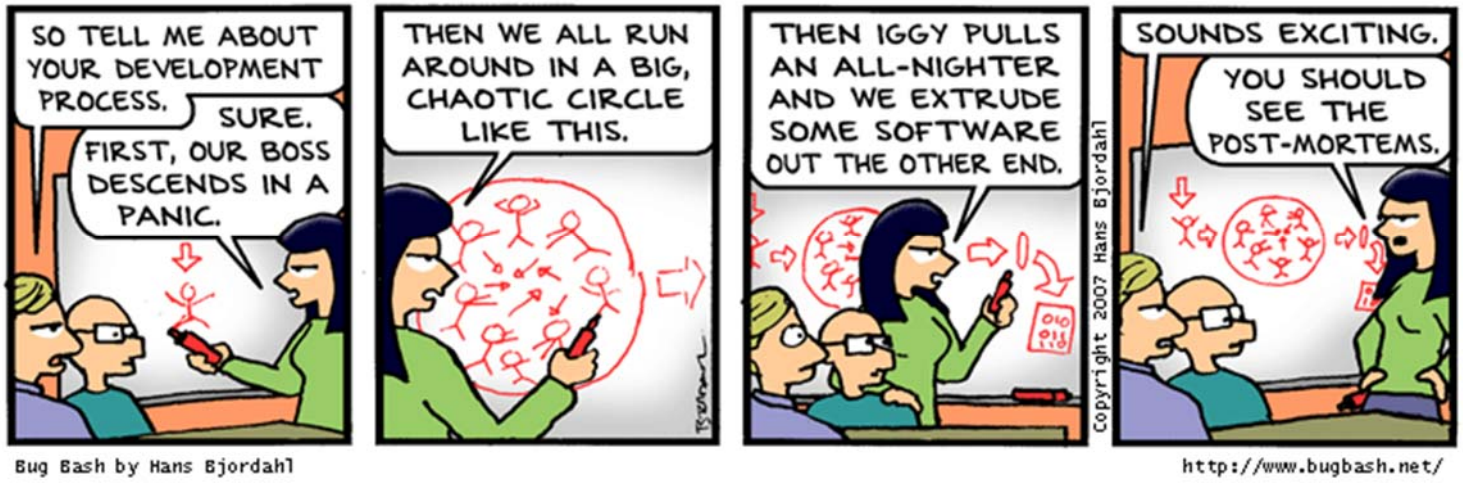
A minimum viable product (MVP) is a concept from Lean Startup that **stresses the impact of learning in new product development**. Eric Ries, defined an MVP as that version of a new product which allows a team to collect the **maximum amount of validated learning about customers with the least effort**. This validated learning comes in the form of whether your customers will actually purchase your product.

A key premise behind the idea of MVP is that **you produce an actual product** (which may be no more than a landing page, or a service with an appearance of automation, but which is fully manual behind the scenes) **that you can offer to customers and observe their actual behavior** with the product or service. Seeing what people actually do with respect to a product is much more reliable than asking people what they would do.

The primary benefit of an MVP is you can gain understanding about your customers' interest in your product without fully developing the product. The sooner you can find out whether your product will appeal to customers, the less effort and expense you spend on a product that will not succeed in the market.

[www.agilealliance.org/agile101/agile-glossary/]

Some development process makes life **easier**, instead of ”desperate hacking”

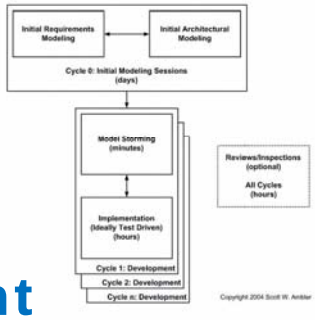


AMDD = agile model driven development

Agile Modeling

AMDD

Agile Model Driven Development (AMDD) is the agile version of Model Driven Development (MDD). The figure below overviews the fundamentals of AMDD:
- At the beginning of a project invest some time to do some initial modeling, particularly to explore the fundamental requirements and to identify a potential architectural approach.
- During development cycles you will typically model for several minutes and then implement for several hours or even days until you need to do some more modeling.
- The AM practices of Model With Others and Collective Ownership are effectively model reviews in process, removing the need for formal reviews.



Values

The Values of AM
- Communication
- Courage
- Feedback
- Humility
- Simplicity

Principles

Core Principles of AM
- Assume Simplicity
- Embrace Change
- Enabling the Next Effort is Your Secondary Goal
- Incremental Change
- Maximize Stakeholder Investment
- Model With a Purpose
- Multiple Models
- Quality Work
- Rapid Feedback
- Software is Your Primary Goal
- Travel Light

Supplementary Principles of AM
- Content Is More Important Than Representation
- Open and Honest Communication

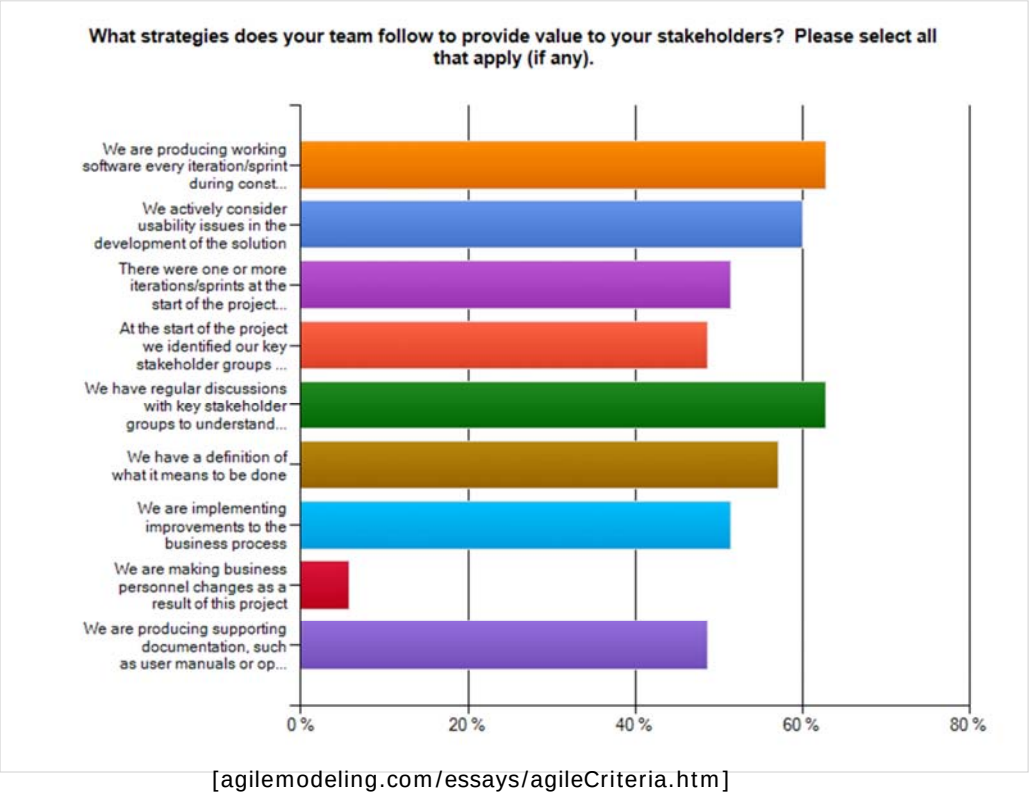
Practices

Core Practices of AM
- Active Stakeholder Participation
- Apply The Right Artifact(s)
- Collective Ownership
- Create Several Models in Parallel
- Create Simple Content
- Depict Models Simply
- Display Models Publicly
- Iterate To Another Artifact
- Model in Small Increments
- Model With Others
- Prove it With Code
- Single Source Information
- Use the Simplest Tools

Supplementary Practices of AM
- Apply Modeling Standards
- Apply Patterns Gently
- Discard Temporary Models
- Formalize Contract Models
- Update Only When It Hurts

Tampereen yliopisto
Tampere University

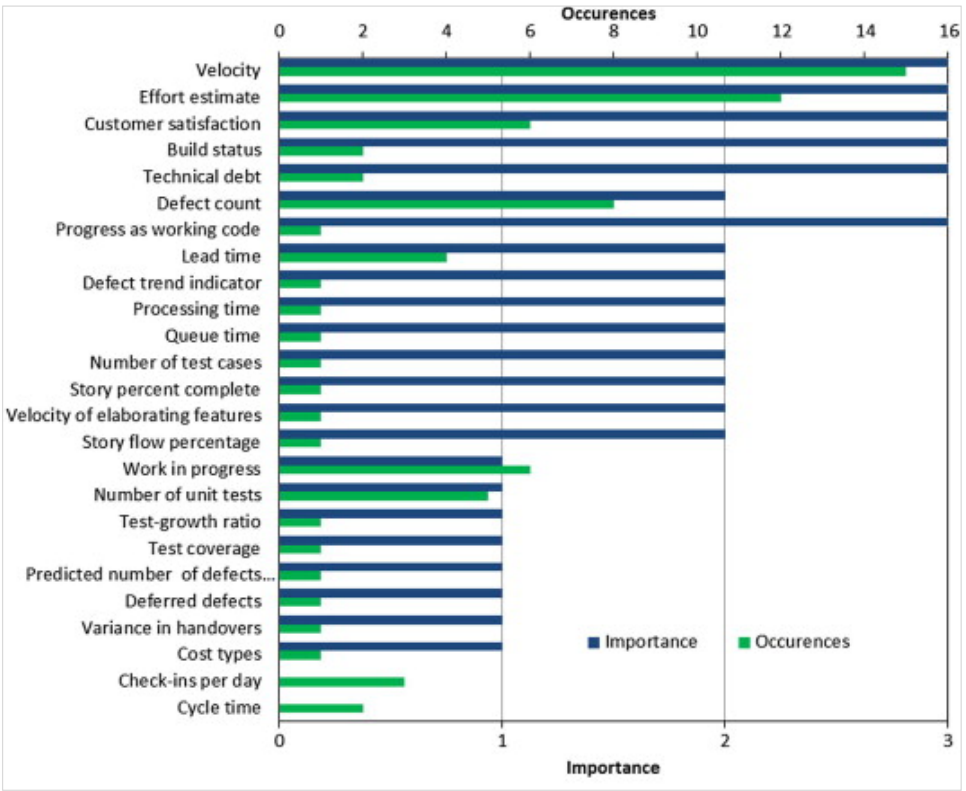
Agile criteria survey



[agilemodeling.com/essays/agileCriteria.htm]

Tampereen yliopisto
Tampere University

Agile metrics survey



[Using metrics in Agile and Lean Software Development, 2015]

Tampereen yliopisto
Tampere University

DevOps values

DevOps is unique to software development methodology because its practice promotes empathy via a culture of **collaboration and communication**, rather than encouraging siloed functionality.

Culture: DevOps seeks to **solve business problems** that arise when people create and manage complex systems. In this regard, DevOps is as much a method for managing human problems as it is a technological solution. A "**people over process over tools**" culture is a central tenet of DevOps. Even with the advent of more innovative tools and advanced computing technology, the process of software development depends on elements of human culture

Automation: DevOps is not just about tools or about automating tasks using software. That said, automation is a core DevOps value, and it is essential to leveraging Agile development practices, including **continuous integration and continuous delivery**. To accommodate continuous releases, DevOps encourages automation. In the DevOps methodology, it's critical to prioritize problem solving that uses automation, and to **make QA everyone's responsibility**.

Measurement: In order to determine if DevOps is continuously improving processes, team members should collect and analyze data. This mandate applies to business-side metrics as well as development, test, and operations metrics.

Sharing: this value is referred as the *loopback* in the DevOps cycle, where stakeholders share ideas and solve problems. **Sharing ideas** helps attract talented people who thrive on feedback to improve. DevOps depends on the principles of continuous improvement and the collaboration those principles foster. Sharing is a core value of DevOps

Tampereen yliopisto
Tampere University

DevOps

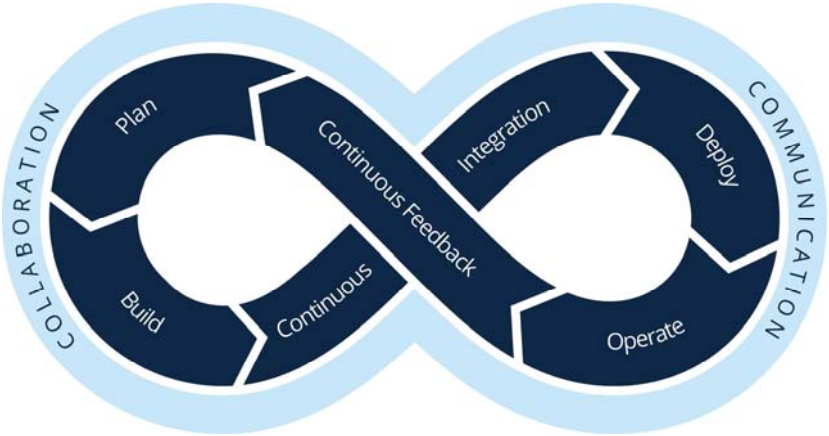
DevOps refers to the **combination** of **software development** (which includes software testing) and **operations methodology** (the values, principles, methods, and practices) to deliver applications and services. DevOps promotes frequent communication and ongoing, real-time collaboration between traditionally disparate workflows of developers and IT operations teams. The DevOps approach to organizing workflows replaces siloed development and IT operations teams with multidisciplinary teams capable of implementing Agile planning and practices, such as continuous integration, continuous delivery, and infrastructure automation.

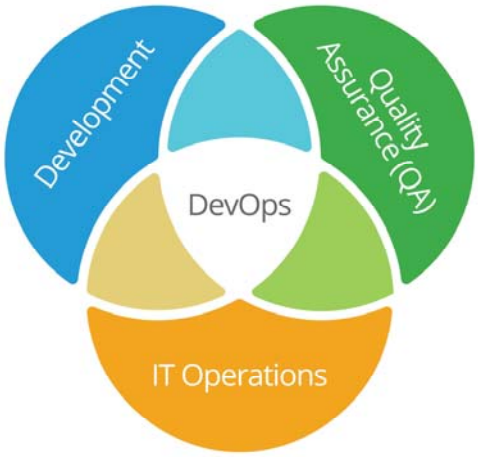
The technology community uses a variety of terms to describe the essence of DevOps. **It is a culture, a movement, a philosophy, a set of practices, and the act of using tools** (software) to automate and improve imperfect methods of managing complex systems.

The DevOps culture is reinforced by the practices it borrows from Agile and Lean principles, with an **added focus on service and quality**. By designing, building, testing, deploying, managing, and operating applications and systems faster and more reliably, DevOps practitioners seek to create value for customers (a profitable competitive advantage) and foster a manageable workflow that places **people over product**.

[www.smartsheet.com/devops]

DevOps is an organizational culture that leverages core principles and technical practices (for example continuous delivery) to optimize application performance and infrastructure and deliver value as a strategic asset.
The DevOps movement is an effort to transform the way various stakeholders interact, communicate, and work together in the software development lifecycle.





[www.smartsheet.com/devops]

DevOps practices

The DevOps methodology leverages key practices and techniques to **streamline software development and operations processes**. **Increasing the frequency and number of daily software deployments** is challenging to teams of all sizes with varying access to resources. To manage the organizational challenges of DevOps, its practitioners leverage practices known as continuous integration (CI) and continuous delivery (CD).

• **Continuous Integration:** CI is the practice of engaging in ongoing testing (leveraging automation) by **merging code development with real-time, problem-seeking tests**. The goals of CI are to **reduce integration problems**, improve quality, reduce time to release, and empower feedback loops that contribute to higher-velocity (daily) deployments. CI leverages comprehensive, automated testing frameworks and continuously addresses problems to keep the system in a working state.

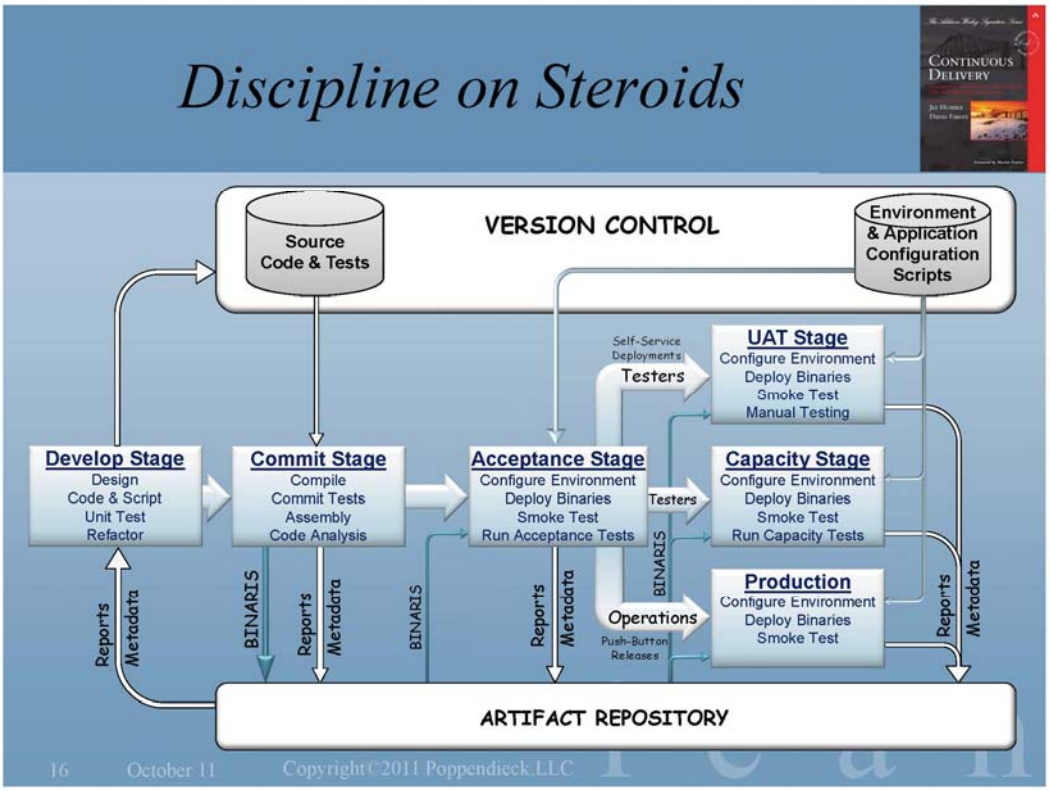
• **Continuous Delivery:** CD is the practice of **frequently building, testing, and releasing code changes** to an (production or testing) environment in small batches after the build stage. The emphasis on **automated testing** (and **automated builds**) for quality assurance capitalizes on the efficiency of successful test automation and is essential to the deployment-ready goal of this practice. You can achieve CD when you synchronize it with the steps required for CI. However, CD does not require the deployment of every build. Continuous deployment deploys every CI build to production.

[https://www.smartsheet.com/devops]

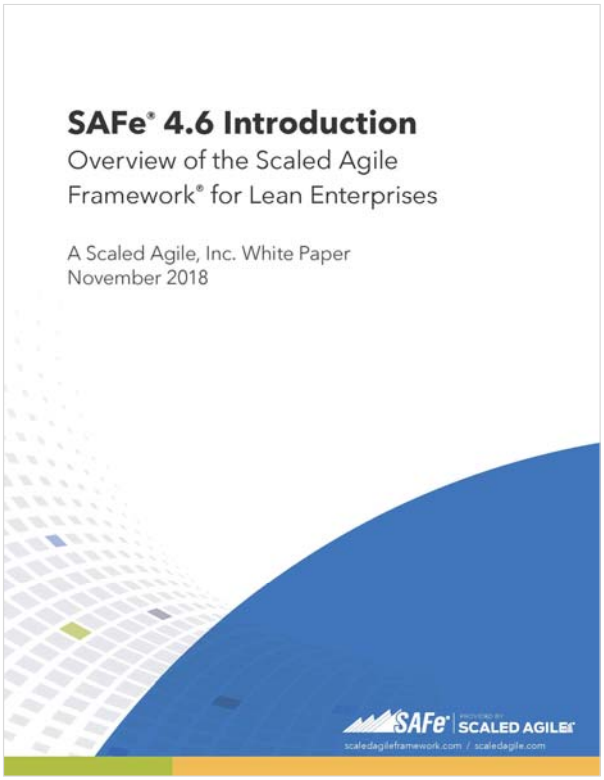
CI = continuous integration

CD = continuous delivery

UAT = user acceptance testing



SAFe = Scaled Agile Framework



SAFe is based on nine immutable, underlying Lean-Agile principles.

These tenets and economic concepts inspire and inform the roles and practices of SAFe, influencing leadership behaviors and decision-making.

1. Take an economic view - An understanding of economics drives decisions.

2. Apply systems thinking - Everyone understands and commits to the common goals of the larger system.

3. Assume variability; preserve options - Decisions are delayed until the last responsible moment.

4. Build incrementally with fast, integrated learning cycles - Cadence-based learning cycles are used to gain knowledge, evaluate alternatives and inform decision-making.

5. Base milestones on objective evaluation of working systems - Progress is measured by objectives measures.

6. Visualize and limit WIP, reduce batch sizes, and manage queue length - Small batches of work, controlled Work in Progress (WIP), and small queues ensures fast flow of value and learning.

7. Apply cadence; synchronize with cross-domain planning - Regular synchronization continually aligns all system builders and ensure all perspectives are understood and resolved.

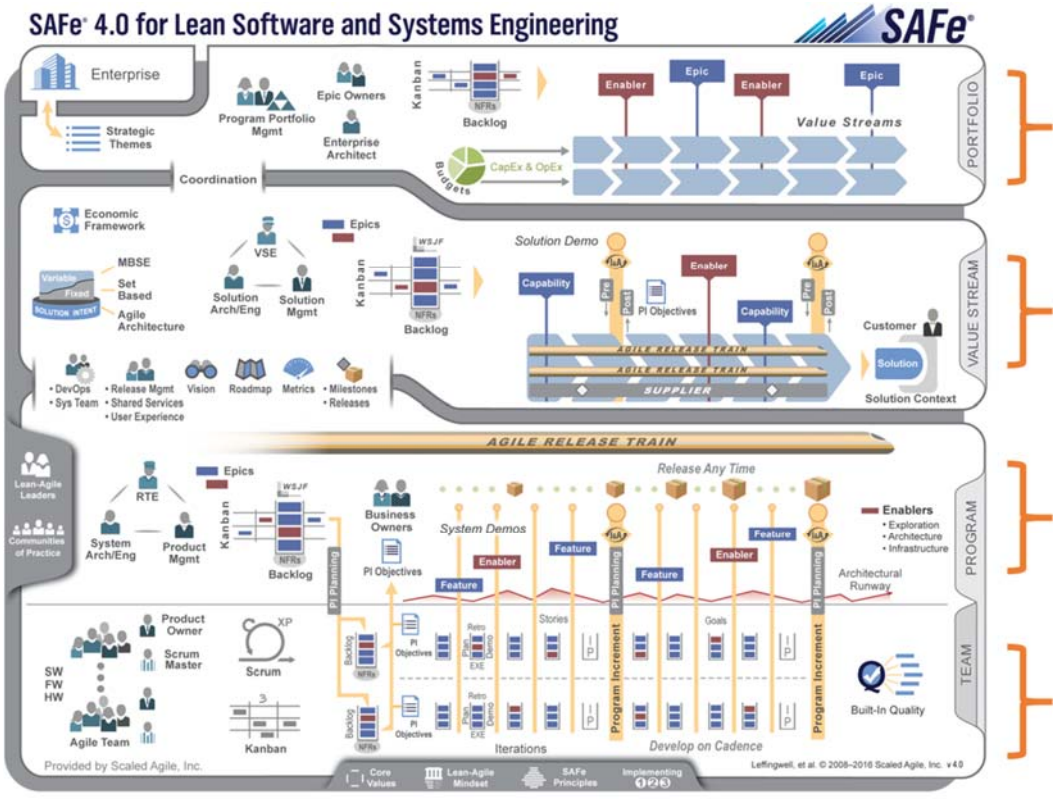
8. Unlock the intrinsic motivation of knowledge workers - Knowledge workers exhibit curiosity and have fundamentally different motivations.

9 . Decentralize decision-making - Autonomy empowers individuals and enhances motivation.

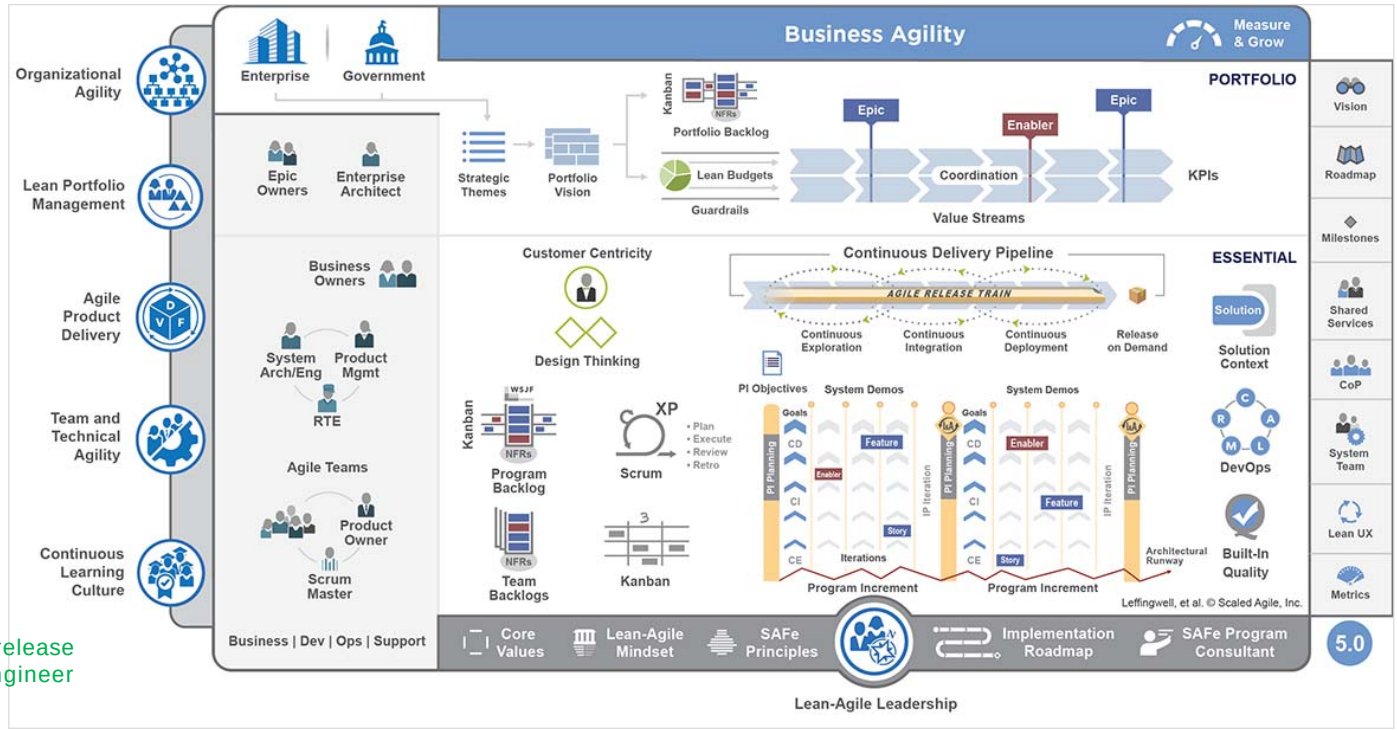
Scaled agile framework.

Four lines.

RTE = release train engineer



Scaled agile framework 5.0



RTE = release train engineer

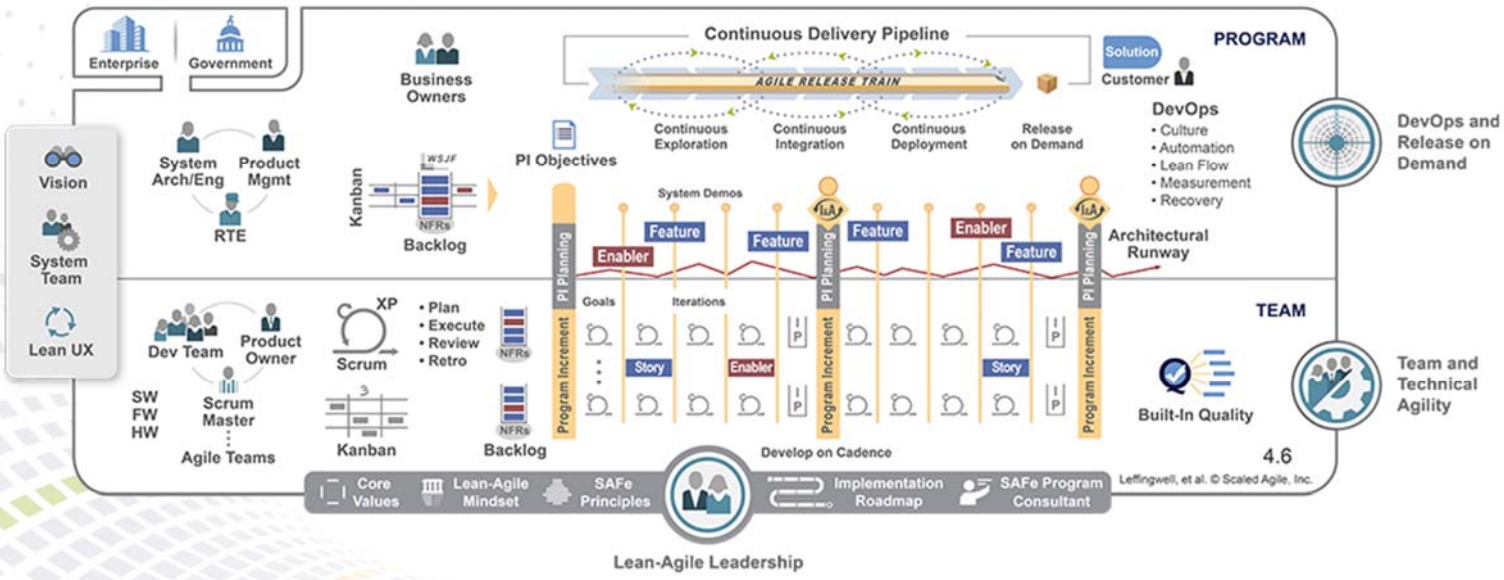
Essential SAFe

The Essential SAFe configuration is the most basic configuration of SAFe. It provides a starting point for implementing SAFe and describes the most critical elements needed to realize the majority of the framework's benefits. It consists of the Team and Program levels, and Foundation.

The Ten Essential Elements

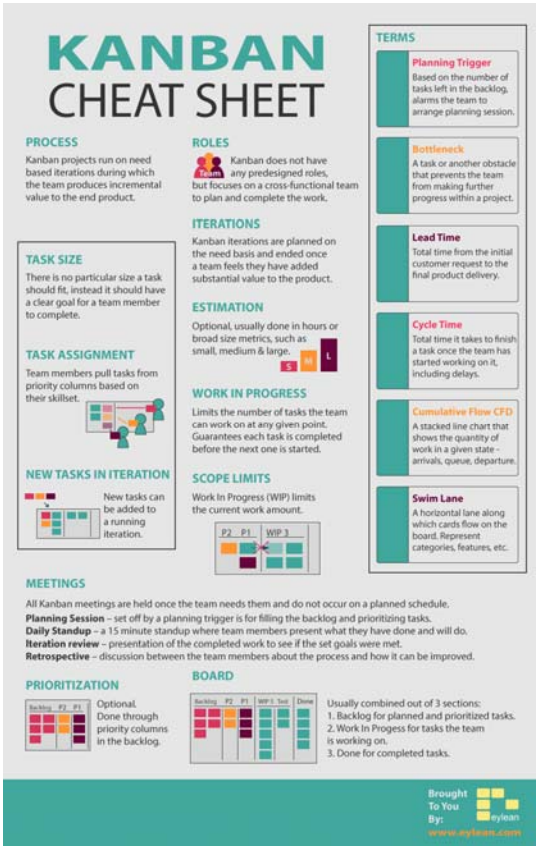
1. Lean-Agile Principles
2. Real Agile Teams and Trains
3. Cadence and synchronization
4. PI planning (PI = program increment)
5. DevOps and releasability
6. System demo
7. Inspect and adapt
8. IP iteration
9. Architectural runway
10. Lean-Agile leadership.

Essential SAFe®



Tampereen yliopisto
Tampere University

kanban



kanban

Kanban term came into existence using the flavors of “visual card,” “signboard,” or “billboard”, “signaling system” to indicate a workflow that limits Work In Progress (WIP). Kanban cards were originally used in Toyota to limit the amount of inventory tied up in “work in progress” on a manufacturing floor. **Kanban not only reduces excess inventory waste, but also the time spent in producing it**. In addition, all of the resources and time freed by the implementation of a Kanban system can be used for future expansions or new opportunities.

The core concept of Kanban includes

- **Visualize Workflow**
  - Split the entire work into defined segments or states, visualized as named columns (lists) on a wall (board).
  - Write each item on a card and put in a column to indicate where the item is in the workflow.
- **Limit WIP**
  - Assign explicit limits to how many items can be in progress at each workflow segment / state. i.e., Work in Progress (WIP) is limited in each workflow state.
- **Measure the Lead Time**
  - Lead Time, also known as cycle time is the average time to complete one item. Measure the Lead Time and optimize the process to make the Lead Time as small and predictable as possible.

[https://www.tutorialspoint.com/kanban/]

kanban

The Kanban Method is a means to design, manage, and improve flow systems for knowledge work. The method also allows organizations to start with their existing workflow and drive evolutionary change. They can do this by visualizing their flow of work, limit **work in progress** (**WIP**) and stop starting and start finishing.

The Kanban Method gets its name from the use of kanban – visual signaling mechanisms to control work in progress for intangible work products.

A general term for systems using the Kanban Method is **flow** – reflecting that work flows continuously through the system instead of being organized into distinct timeboxes.

Kanban can be used in any knowledge work setting, and is **particularly applicable in situations where work arrives in an unpredictable fashion** and/or when you want to deploy work as soon as it is ready, rather than waiting for other work items.

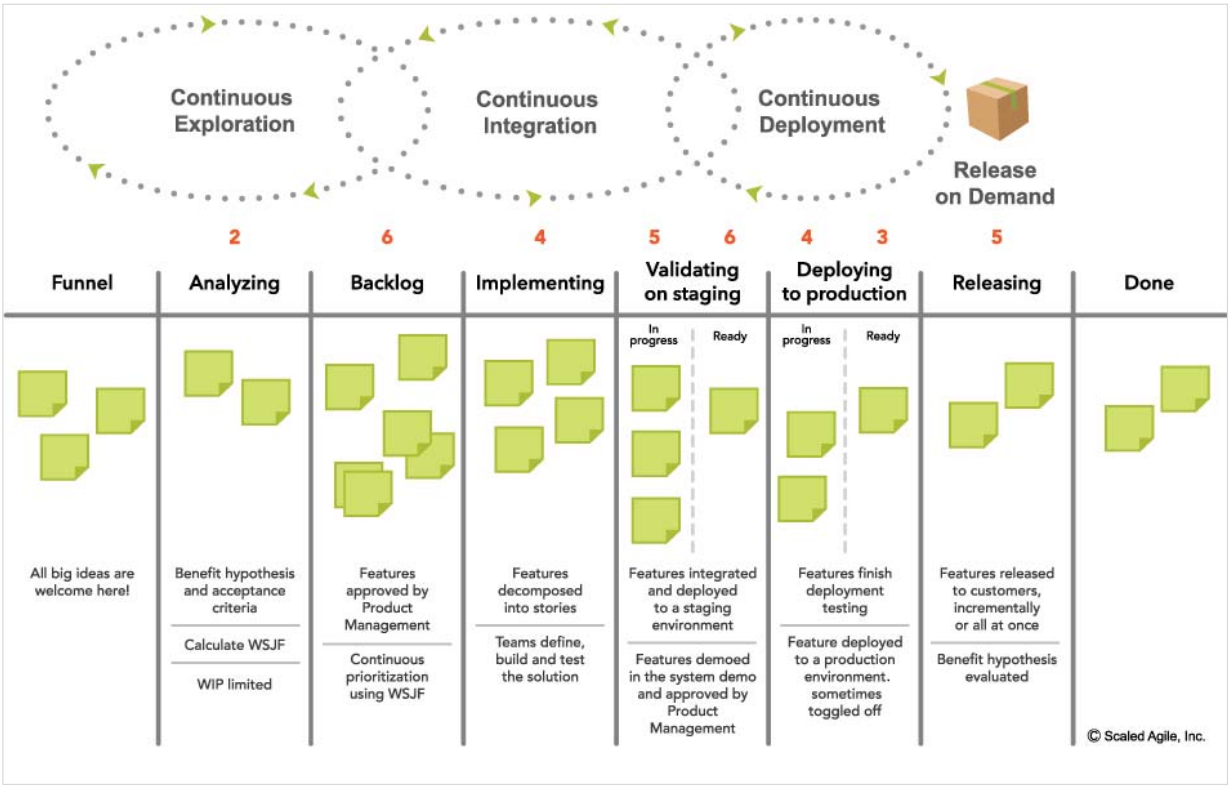
[www.agilealliance.org/glossary/]

Tampereen yliopisto
Tampere University

The program Kanban facilitates the flow of features through the continuous delivery pipeline.

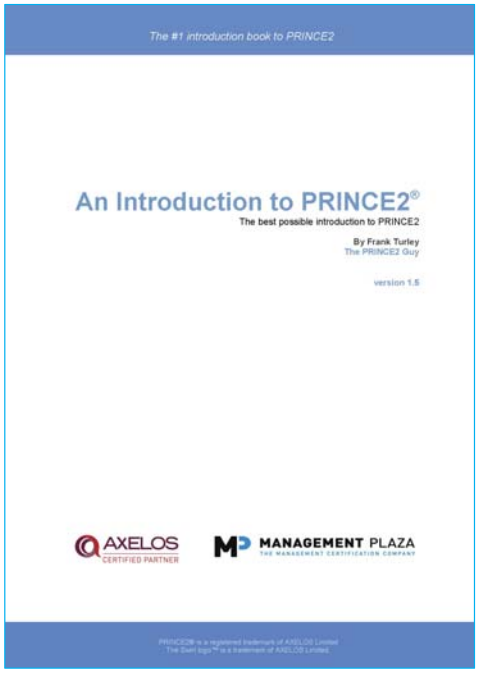
WSJF = weighted shortest job first

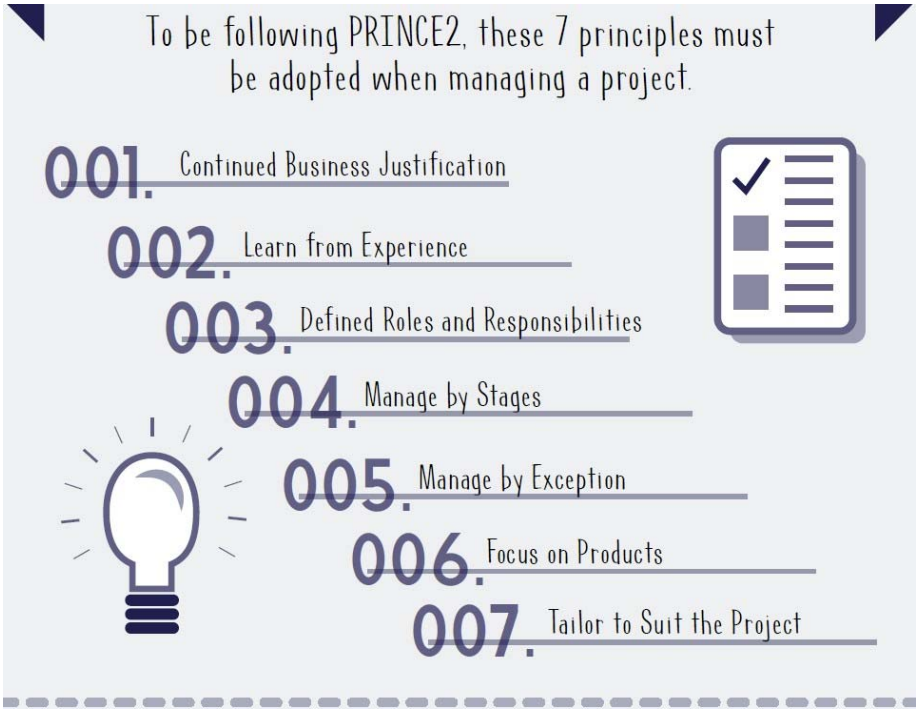
WIP = work in process



Tampereen yliopisto
Tampere University

PRINCE2, PRojects IN Controlled Environments v.2





[https://www.itonlinelearning.com/prince2-cheat-sheet/]

Tampereen yliopisto
Tampere University

PRINCE2

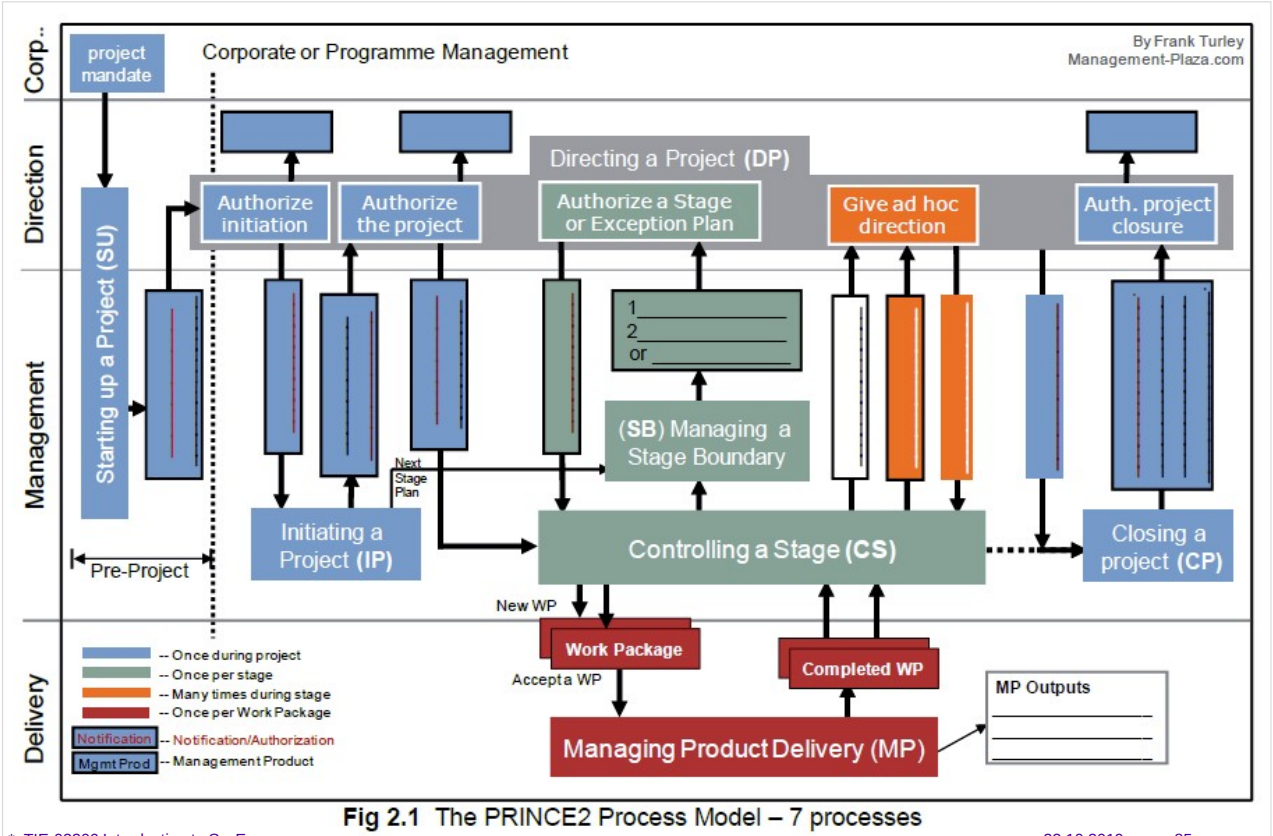


Fig 2.1 The PRINCE2 Process Model – 7 processes

Tampereen yliopisto
Tampere University

The most popular: Scrum-BUT

The most used Scrum method in Finland (and in the world) is **Scrum-BUT**.

If you ask some company ”Are you using Scrum ?”, they will usually reply ”**Yes, we are using Scrum… but we have made a few shortcuts and modifications**.”

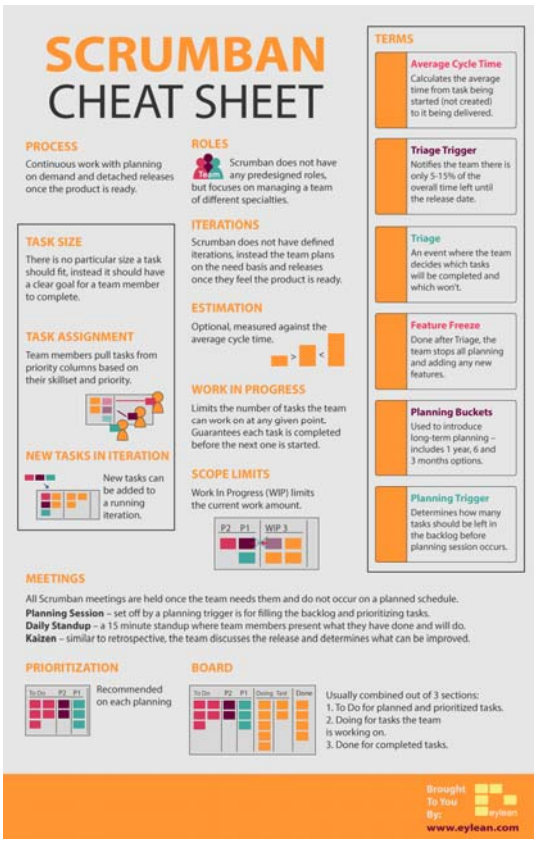
So, **Scrum-BUT means modified Scrum**. Adapt ”best practices” from all public sources you see or hear, and **make your own method combination which suits best your work**.

When James Coplien had Scrum Master and Product Owner traing courses at TUT, he said that ”**If you are using agile / Scrum by the book, you are not agile**”.

DIFFERENCE BETWEEN	
SCRUM	**KANBAN**
Timeboxed iterations prescribed.	Timeboxed iterations optional. Can have separate cadences for planning, release, and process improvement. Can be event driven instead of timeboxed.
Team commits to a specific amount of work for this iteration.	Commitment optional.
Uses Velocity as default metric for planning and process improvement.	Uses Lead time as default metric for planning and process improvement.
Cross-functional teams prescribed.	Cross-functional teams optional. Specialist teams allowed.
Items must be broken down so they can be completed within 1 sprint.	No particular item size is prescribed.
Burndown chart prescribed.	No particular type of diagram is prescribed.
WIP limited indirectly (per sprint).	WIP limited directly (per workflow state).
Estimation prescribed.	Estimation optional.
Can not add items to ongoing iteration.	Can add new items whenever capacity is available.
A sprint backlog is owned by one specific team.	A Kanban board may be shared by multiple teams or individuals.
Prescribes 3 roles (PO/SM/Team).	Doesn't prescribe any roles.
A Scrum board is reset between each sprint.	A Kanban board is persistent.
Prescribes a prioritized product backlog.	Prioritization is optional.

[www.cuelogic.com]

Scrumban



Highlights - What to remember

- there are many many methods, no one fits all sw dev projects
- ”Aim for best practices and standard conventions.”
- agile means taking customer’s changes into account all the time during the project
- agile documentation is minimum, for user’s and maintenance needs; code comments, Product Backlog, some Test log/report evidence, perhaps User Manual, Maintenance quide,…